The Parma Polyhedra Library Developer's Manual* (version 0.11.2)

Roberto Bagnara†
Patricia M. Hill‡
Enea Zaffanella§

based on previous work also by

Elisa Ricci

and

Sara Bonini
Andrea Pescetti
Angela Stazzone
Tatiana Zolo

February 27, 2011

*This work has been partly supported by: University of Parma's FIL scientific research project (ex 60%) "Pure and Applied Mathematics"; MURST project "Automatic Program Certification by Abstract Interpretation"; MURST project "Abstract Interpretation, Type Systems and Control-Flow Analysis"; MURST project "Automatic Aggregate- and Number-Reasoning for Computing: from Decision Algorithms to Constraint Programming with Multisets, Sets, and Maps"; MURST project "Constraint Based Verification of Reactive Systems"; MURST project "Abstract Interpretation: Design and Applications"; EPSRC project "Numerical Domains for Software Analysis"; EPSRC project "Geometric Abstractions for Scalable Program Analyzers".

†bagnara@cs.unipr.it, Department of Mathematics, University of Parma, Italy.
‡hill@comp.leeds.ac.uk, School of Computing, University of Leeds, U.K.
§zaffanella@cs.unipr.it, Department of Mathematics, University of Parma, Italy.

Copyright © 2001–2010 Roberto Bagnara (bagnara@cs.unipr.it).

This document describes the Parma Polyhedra Library (PPL).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

The PPL is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. A copy of the license is included in the section entitled "GNU GENERAL PUBLIC LICENSE".

The PPL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For the most up-to-date information see the Parma Polyhedra Library site:

http://www.cs.unipr.it/ppl/

Contents

1 General Information on the PPL **1**

1.1 The Main Features . . . 1
1.2 Upward Approximation . . . 6
1.3 Approximating Integers . . . 6
1.4 Convex Polyhedra . . . 8
1.5 Representations of Convex Polyhedra . . . 9
1.6 Operations on Convex Polyhedra . . . 13
1.7 Intervals and Boxes . . . 20
1.8 Weakly-Relational Shapes . . . 20
1.9 Rational Grids . . . 22
1.10 Operations on Rational Grids . . . 24
1.11 The Powerset Construction . . . 27
1.12 Operations on the Powerset Construction . . . 28
1.13 The Pointset Powerset Domain . . . 28
1.14 Using the Library . . . 30
1.15 Bibliography . . . 32
1.16 Further Notation and Terminology . . . 39
1.17 Homogeneous Systems . . . 41
1.18 Integers Represented by Floating Point Numbers . . . 45

2 GNU General Public License **47**

3 GNU Free Documentation License **56**

4 Module Index **61**

4.1 Modules 61

5 Namespace Index **62**

5.1 Namespace List 62

6 Class Index **63**

6.1 Class Hierarchy 63

7 Class Index **71**

7.1 Class List 71

8 File Index **81**

8.1 File List 81

9 Module Documentation **89**

9.1 C++ Language Interface 89

10 Namespace Documentation **104**

10.1 Parma_Polyhedra_Library Namespace Reference 104

10.2 Parma_Polyhedra_Library::Checked Namespace Reference 166

10.3 Parma_Polyhedra_Library::Implementation Namespace Reference 225

10.4 Parma_Polyhedra_Library::Implementation::Octagonal_Shapes Namespace Reference . . 226

10.5 Parma_Polyhedra_Library::Implementation::Pointset_Powersets Namespace Reference . . 226

10.6 Parma_Polyhedra_Library::Implementation::Termination Namespace Reference 226

10.7 Parma_Polyhedra_Library::Interfaces Namespace Reference 231

10.8 Parma_Polyhedra_Library::Interfaces::C Namespace Reference 232

10.9 Parma_Polyhedra_Library::Interfaces::Java Namespace Reference 232

10.10 Parma_Polyhedra_Library::Interfaces::OCaml Namespace Reference 233

10.11 Parma_Polyhedra_Library::Interfaces::Prolog Namespace Reference 233

10.12 Parma_Polyhedra_Library::Interfaces::Prolog::Ciao Namespace Reference 233

10.13 Parma_Polyhedra_Library::Interfaces::Prolog::GNU Namespace Reference 234

10.14 Parma_Polyhedra_Library::Interfaces::Prolog::SICStus Namespace Reference 234

10.15 Parma_Polyhedra_Library::Interfaces::Prolog::SWI Namespace Reference 234

10.16 Parma_Polyhedra_Library::Interfaces::Prolog::XSB Namespace Reference 234

10.17 Parma_Polyhedra_Library::Interfaces::Prolog::YAP Namespace Reference 234

10.18 Parma_Polyhedra_Library::Interval_NS Namespace Reference 235

10.19 Parma_Polyhedra_Library::IO_Operators Namespace Reference 240
10.20 std Namespace Reference 241

11 Class Documentation **241**

11.1 Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any Struct Reference 241
11.2 Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U > Class Template Reference 242
11.3 Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter Class Reference 251
11.4 Parma_Polyhedra_Library::BD_Shape< T > Class Template Reference 258
11.5 Parma_Polyhedra_Library::BHRZ03_Certificate Class Reference 337
11.6 Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign > Class Template Reference 342
11.7 Parma_Polyhedra_Library::Bit_Matrix Class Reference 344
11.8 Parma_Polyhedra_Library::Bit_Row Class Reference 354
11.9 Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than Struct Reference 363
11.10 Parma_Polyhedra_Library::Bool< b > Struct Template Reference 364
11.11 Parma_Polyhedra_Library::Boundary_Value< T, Enable > Struct Template Reference . . 365
11.12 Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type > Struct Template Reference 366
11.13 Parma_Polyhedra_Library::Box< ITV > Class Template Reference 366
11.14 Parma_Polyhedra_Library::C_Polyhedron Class Reference 436
11.15 Parma_Polyhedra_Library::Check_Overflow_Policy< T > Struct Template Reference . . 444
11.16 Parma_Polyhedra_Library::Checked_Number< T, Policy > Class Template Reference . . 446
11.17 Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T > Struct Template Reference 475
11.18 Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient > Struct Template Reference 480
11.19 Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer > Struct Template Reference 480
11.20 Parma_Polyhedra_Library::BHRZ03_Certificate::Compare Struct Reference 481
11.21 Parma_Polyhedra_Library::H79_Certificate::Compare Struct Reference 482
11.22 Parma_Polyhedra_Library::Grid_Certificate::Compare Struct Reference 482
11.23 Parma_Polyhedra_Library::Variable::Compare Struct Reference 483
11.24 Parma_Polyhedra_Library::Compile_Time_Check< true > Struct Template Reference . . 484
11.25 Parma_Polyhedra_Library::Congruence Class Reference 484
11.26 Parma_Polyhedra_Library::Congruence_System Class Reference 504
11.27 Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 > Class Template Reference 523
11.28 Parma_Polyhedra_Library::Matrix::const_iterator Class Reference 525
11.29 Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator Class Reference 529

11.30Parma_Polyhedra_Library::Constraint_System::const_iterator Class Reference 534
11.31Parma_Polyhedra_Library::Congruence_System::const_iterator Class Reference 539
11.32Parma_Polyhedra_Library::Generator_System::const_iterator Class Reference 544
11.33Parma_Polyhedra_Library::Grid_Generator_System::const_iterator Class Reference . . . 549
11.34Parma_Polyhedra_Library::const_iterator_to_const< Container > Class Template Reference 554
11.35Parma_Polyhedra_Library::Constant< v, prefer_signed > Struct Template Reference . . . 560
11.36Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable > Struct Template Reference 561
11.37Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type > Struct Template Reference 562
11.38Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type > Struct Template Reference 562
11.39Parma_Polyhedra_Library::Constraint Class Reference 563
11.40Parma_Polyhedra_Library::Constraint_System Class Reference 588
11.41Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 > Class Template Reference 606
11.42Parma_Polyhedra_Library::DB_Matrix< T > Class Template Reference 608
11.43Parma_Polyhedra_Library::DB_Row< T > Class Template Reference 619
11.44Parma_Polyhedra_Library::DB_Row_Impl_Handler< T > Class Template Reference . . . 632
11.45Parma_Polyhedra_Library::Determinate< PSET > Class Template Reference 635
11.46Parma_Polyhedra_Library::Domain_Product< D1, D2 > Class Template Reference 644
11.47Parma_Polyhedra_Library::Enable_If< b, T > Struct Template Reference 646
11.48Parma_Polyhedra_Library::Enable_If< true, T > Struct Template Reference 646
11.49Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T > Struct Template Reference . . 647
11.50Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp > Struct Template Reference 648
11.51Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type > Struct Template Reference 648
11.52Parma_Polyhedra_Library::Extended_Number_Policy Struct Reference 650
11.53Parma_Polyhedra_Library::False Struct Reference 652
11.54Parma_Polyhedra_Library::Fit< T, v, Enable > Struct Template Reference 653
11.55Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type > Struct Template Reference 655
11.56Parma_Polyhedra_Library::Row::Flags Class Reference 655
11.57Parma_Polyhedra_Library::Linear_Row::Flags Class Reference 661
11.58Parma_Polyhedra_Library::Float< T > Class Template Reference 668
11.59Parma_Polyhedra_Library::Checked::Float_2exp Struct Reference 670

11.60Parma_Polyhedra_Library::float_ibm_double Struct Reference 670
11.61Parma_Polyhedra_Library::float_ibm_single Struct Reference 671
11.62Parma_Polyhedra_Library::float_ieee754_double Struct Reference 676
11.63Parma_Polyhedra_Library::float_ieee754_half Struct Reference 681
11.64Parma_Polyhedra_Library::float_ieee754_quad Struct Reference 685
11.65Parma_Polyhedra_Library::float_ieee754_single Struct Reference 690
11.66Parma_Polyhedra_Library::float_intel_double_extended Struct Reference 695
11.67Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > > Struct Template Reference 700
11.68Parma_Polyhedra_Library::Generator Class Reference 700
11.69Parma_Polyhedra_Library::Generator_System Class Reference 726
11.70Parma_Polyhedra_Library::GMP_Integer Class Reference 745
11.71Parma_Polyhedra_Library::Grid Class Reference 749
11.72Parma_Polyhedra_Library::Grid_Certificate Class Reference 832
11.73Parma_Polyhedra_Library::Grid_Generator Class Reference 835
11.74Parma_Polyhedra_Library::Grid_Generator_System Class Reference 857
11.75Parma_Polyhedra_Library::H79_Certificate Class Reference 876
11.76Parma_Polyhedra_Library::Has_OK< T, Enable > Struct Template Reference 879
11.77Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::∗)() const,&T::OK >::type > Struct Template Reference 881
11.78Parma_Polyhedra_Library::Row_Impl_Handler::Impl Class Reference 882
11.79Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl Class Reference 890
11.80Parma_Polyhedra_Library::Init Class Reference 898
11.81Parma_Polyhedra_Library::Interval< Boundary, Info > Class Template Reference 900
11.82Parma_Polyhedra_Library::Interval_Base Struct Reference 930
11.83Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > Class Template Reference . 930
11.84Parma_Polyhedra_Library::Interval_Info_Null< Policy > Class Template Reference . . . 937
11.85Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy > Class Template Reference 941
11.86Parma_Polyhedra_Library::Interval_Restriction_Integer< Base > Class Template Reference943
11.87Parma_Polyhedra_Library::Interval_Restriction_Integer_Base Class Reference 945
11.88Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base > Class Template Reference 946
11.89Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base Class Reference . 949
11.90Parma_Polyhedra_Library::Interval_Restriction_None< Base > Class Template Reference 950
11.91Parma_Polyhedra_Library::Interval_Restriction_None_Base Class Reference 951
11.92Parma_Polyhedra_Library::Is_Checked< T > Struct Template Reference 952
11.93Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > > Struct Template Reference 953

11.94 Parma_Polyhedra_Library::Is_Interval< T > Struct Template Reference 955
11.95 Parma_Polyhedra_Library::Is_Native< T > Struct Template Reference 956
11.96 Parma_Polyhedra_Library::Is_Native< char > Struct Template Reference 957
11.97 Parma_Polyhedra_Library::Is_Native< mpq_class > Struct Template Reference 959
11.98 Parma_Polyhedra_Library::Is_Native< mpz_class > Struct Template Reference 960
11.99 Parma_Polyhedra_Library::Is_Native< signed char > Struct Template Reference 962
11.100 Parma_Polyhedra_Library::Is_Native< signed int > Struct Template Reference 963
11.101 Parma_Polyhedra_Library::Is_Native< signed long > Struct Template Reference 965
11.102 Parma_Polyhedra_Library::Is_Native< signed long long > Struct Template Reference . . 966
11.103 Parma_Polyhedra_Library::Is_Native< signed short > Struct Template Reference 968
11.104 Parma_Polyhedra_Library::Is_Native< unsigned char > Struct Template Reference 969
11.105 Parma_Polyhedra_Library::Is_Native< unsigned int > Struct Template Reference 971
11.106 Parma_Polyhedra_Library::Is_Native< unsigned long > Struct Template Reference 972
11.107 Parma_Polyhedra_Library::Is_Native< unsigned long long > Struct Template Reference . 974
11.108 Parma_Polyhedra_Library::Is_Native< unsigned short > Struct Template Reference . . . 975
11.109 Parma_Polyhedra_Library::Is_Native_Or_Checked< T > Struct Template Reference . . . 977
11.110 Parma_Polyhedra_Library::Is_Same< T1, T2 > Struct Template Reference 978
11.111 Parma_Polyhedra_Library::Is_Same< T, T > Struct Template Reference 979
11.112 Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived > Struct Template Reference 981
11.113 Parma_Polyhedra_Library::Is_Singleton< T, Enable > Struct Template Reference 983
11.114 Parma_Polyhedra_Library::Is_Special< T > Struct Template Reference 985
11.115 Parma_Polyhedra_Library::Is_Special< Minus_Infinity > Struct Template Reference . . . 986
11.116 Parma_Polyhedra_Library::Is_Special< Not_A_Number > Struct Template Reference . . 988
11.117 Parma_Polyhedra_Library::Is_Special< Plus_Infinity > Struct Template Reference 989
11.118 Parma_Polyhedra_Library::iterator_to_const< Container > Class Template Reference . . 991
11.119 Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp > Struct Template Reference 997
11.120 Parma_Polyhedra_Library::Checked::Larger< char > Struct Template Reference 997
11.121 Parma_Polyhedra_Library::Checked::Larger< signed char > Struct Template Reference . 999
11.122 Parma_Polyhedra_Library::Checked::Larger< signed int > Struct Template Reference . . 1001
11.123 Parma_Polyhedra_Library::Checked::Larger< signed long > Struct Template Reference . 1002
11.124 Parma_Polyhedra_Library::Checked::Larger< signed long long > Struct Template Reference 1004
11.125 Parma_Polyhedra_Library::Checked::Larger< signed short > Struct Template Reference . 1005
11.126 Parma_Polyhedra_Library::Checked::Larger< unsigned char > Struct Template Reference 1007
11.127 Parma_Polyhedra_Library::Checked::Larger< unsigned int > Struct Template Reference . 1009
11.128 Parma_Polyhedra_Library::Checked::Larger< unsigned long > Struct Template Reference 1010

11.129 Parma_Polyhedra_Library::Checked::Larger< unsigned long long > Struct Template Reference . . . 1012
11.130 Parma_Polyhedra_Library::Checked::Larger< unsigned short > Struct Template Reference 1013
11.131 Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS > Class Template Reference . . . 1015
11.132 Parma_Polyhedra_Library::Linear_Expression Class Reference . . . 1017
11.133 Parma_Polyhedra_Library::Linear_Row Class Reference . . . 1039
11.134 Parma_Polyhedra_Library::Linear_System Class Reference . . . 1056
11.135 Parma_Polyhedra_Library::Matrix Class Reference . . . 1078
11.136 Parma_Polyhedra_Library::maybe_assign_struct< To, From > Struct Template Reference 1097
11.137 Parma_Polyhedra_Library::maybe_assign_struct< Type, Type > Struct Template Reference 1097
11.138 Parma_Polyhedra_Library::Minus_Infinity Struct Reference . . . 1098
11.139 Parma_Polyhedra_Library::MIP_Problem Class Reference . . . 1098
11.140 Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > > Struct Template Reference . . . 1131
11.141 Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > Struct Template Reference . . . 1132
11.142 Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > > Struct Template Reference . . . 1133
11.143 Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > Struct Template Reference . . . 1134
11.144 Parma_Polyhedra_Library::NNC_Polyhedron Class Reference . . . 1135
11.145 Parma_Polyhedra_Library::PIP_Solution_Node::No_Constraints Struct Reference . . . 1143
11.146 Parma_Polyhedra_Library::No_Reduction< D1, D2 > Class Template Reference . . . 1143
11.147 Parma_Polyhedra_Library::Not_A_Number Struct Reference . . . 1144
11.148 Parma_Polyhedra_Library::Checked::number_struct Struct Reference . . . 1145
11.149 Parma_Polyhedra_Library::Octagonal_Shape< T > Class Template Reference . . . 1146
11.150 Parma_Polyhedra_Library::OR_Matrix< T > Class Template Reference . . . 1227
11.151 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > Class Template Reference . . . 1245
11.152 Parma_Polyhedra_Library::PIP_Decision_Node Class Reference . . . 1289
11.153 Parma_Polyhedra_Library::PIP_Problem Class Reference . . . 1297
11.154 Parma_Polyhedra_Library::PIP_Solution_Node Class Reference . . . 1321
11.155 Parma_Polyhedra_Library::PIP_Tree_Node Class Reference . . . 1333
11.156 Parma_Polyhedra_Library::Plus_Infinity Struct Reference . . . 1347
11.157 Parma_Polyhedra_Library::Pointset_Powerset< PSET > Class Template Reference . . . 1347
11.158 Parma_Polyhedra_Library::Poly_Con_Relation Class Reference . . . 1399
11.159 Parma_Polyhedra_Library::Poly_Gen_Relation Class Reference . . . 1407
11.160 Parma_Polyhedra_Library::Polyhedron Class Reference . . . 1414

11.161Parma_Polyhedra_Library::Powerset< D > Class Template Reference 1514
11.162Parma_Polyhedra_Library::Interval_NS::Property Struct Reference 1534
11.163Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U > Class Template Reference 1536
11.164Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation Struct Reference 1539
11.165Parma_Polyhedra_Library::Rectilinear_Distance_Specialization< Temp > Struct Template Reference 1541
11.166Parma_Polyhedra_Library::Recycle_Input Struct Reference 1541
11.167Parma_Polyhedra_Library::Determinate< PSET >::Rep Class Reference 1542
11.168Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > Struct Template Reference 1547
11.169Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > Struct Template Reference 1548
11.170Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > Struct Template Reference 1549
11.171Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > Struct Template Reference 1550
11.172Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > Struct Template Reference 1551
11.173Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > Struct Template Reference 1552
11.174Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > Struct Template Reference 1553
11.175Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > Struct Template Reference 1554
11.176Parma_Polyhedra_Library::Row Class Reference 1555
11.177Parma_Polyhedra_Library::Row_Impl_Handler Class Reference 1568
11.178Parma_Polyhedra_Library::Linear_System::Row_Less_Than Struct Reference 1571
11.179Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 > Struct Template Reference 1571
11.180Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T > Struct Template Reference 1573
11.181Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable > Struct Template Reference 1574
11.182Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type > Struct Template Reference 1576

11.183 Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy Struct Reference . . . 1576
11.184 Parma_Polyhedra_Library::Scalar_Products Class Reference . . . 1577
11.185 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type > Struct Template Reference . . . 1585
11.186 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char > Struct Template Reference . . . 1586
11.187 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float > Struct Template Reference . . . 1586
11.188 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char > Struct Template Reference . . . 1587
11.189 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int > Struct Template Reference . . . 1587
11.190 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long > Struct Template Reference . . . 1588
11.191 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short > Struct Template Reference . . . 1588
11.192 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char > Struct Template Reference . . . 1589
11.193 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int > Struct Template Reference . . . 1589
11.194 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long > Struct Template Reference . . . 1590
11.195 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long > Struct Template Reference . . . 1591
11.196 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short > Struct Template Reference . . . 1591
11.197 Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 > Class Template Reference . . . 1592
11.198 Parma_Polyhedra_Library::Simple_Restriction_Integer Class Reference . . . 1593
11.199 Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T > Struct Template Reference . . . 1595
11.200 Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > > Struct Template Reference . . . 1597
11.201 Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > > Struct Template Reference . . . 1598
11.202 Parma_Polyhedra_Library::Smash_Reduction< D1, D2 > Class Template Reference . . . 1599
11.203 Parma_Polyhedra_Library::Grid::Status Class Reference . . . 1600
11.204 Parma_Polyhedra_Library::Box< ITV >::Status Class Reference . . . 1610
11.205 Parma_Polyhedra_Library::BD_Shape< T >::Status Class Reference . . . 1615
11.206 Status Class Reference . . . 1622
11.207 Parma_Polyhedra_Library::Polyhedron::Status Class Reference . . . 1630
11.208 Parma_Polyhedra_Library::Octagonal_Shape< T >::Status Class Reference . . . 1644

11.209 Parma_Polyhedra_Library::stdiobuf Class Reference 1649
11.210 Parma_Polyhedra_Library::PIP_Solution_Node::Tableau Struct Reference 1653
11.211 Parma_Polyhedra_Library::TConstant< T, v > Struct Template Reference 1658
11.212 Parma_Polyhedra_Library::Throwable Class Reference 1659
11.213 Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign Class Reference . . 1661
11.214 Parma_Polyhedra_Library::True Struct Reference 1663
11.215 Parma_Polyhedra_Library::Variable Class Reference 1664
11.216 Parma_Polyhedra_Library::Variables_Set Class Reference 1671
11.217 Parma_Polyhedra_Library::Weightwatch_Traits Struct Reference 1676
11.218 Parma_Polyhedra_Library::Widening_Function< PSET > Class Template Reference . . . 1678
11.219 Parma_Polyhedra_Library::Linear_System::With_Pending Struct Reference 1680

12 File Documentation **1680**

12.1 algorithms.hh File Reference 1680
12.2 BD_Shape.cc File Reference 1681
12.3 BD_Shape.defs.hh File Reference 1681
12.4 BD_Shape.inlines.hh File Reference 1683
12.5 BD_Shape.templates.hh File Reference 1684
12.6 BD_Shape.types.hh File Reference 1686
12.7 BHRZ03_Certificate.cc File Reference 1686
12.8 BHRZ03_Certificate.defs.hh File Reference 1686
12.9 BHRZ03_Certificate.inlines.hh File Reference 1687
12.10 BHRZ03_Certificate.types.hh File Reference 1688
12.11 Bit_Matrix.cc File Reference 1688
12.12 Bit_Matrix.defs.hh File Reference 1689
12.13 Bit_Matrix.inlines.hh File Reference 1690
12.14 Bit_Matrix.types.hh File Reference 1691
12.15 Bit_Row.cc File Reference 1691
12.16 Bit_Row.defs.hh File Reference 1691
12.17 Bit_Row.inlines.hh File Reference 1692
12.18 Bit_Row.types.hh File Reference 1694
12.19 Box.cc File Reference 1694
12.20 Box.defs.hh File Reference 1694
12.21 Box.inlines.hh File Reference 1696
12.22 Box.templates.hh File Reference 1697
12.23 Box.types.hh File Reference 1699
12.24 C_Polyhedron.cc File Reference 1699

12.25C_Polyhedron.defs.hh File Reference 1701
12.26C_Polyhedron.inlines.hh File Reference 1702
12.27C_Polyhedron.types.hh File Reference 1703
12.28checked.cc File Reference 1703
12.29checked.defs.hh File Reference 1704
12.30checked.inlines.hh File Reference 1719
12.31checked_float.inlines.hh File Reference 1723
12.32checked_int.inlines.hh File Reference 1727
12.33checked_mpq.inlines.hh File Reference 1733
12.34checked_mpz.inlines.hh File Reference 1737
12.35Checked_Number.cc File Reference 1740
12.36Checked_Number.defs.hh File Reference 1740
12.37Checked_Number.inlines.hh File Reference 1744
12.38Checked_Number.types.hh File Reference 1748
12.39Coefficient.cc File Reference 1749
12.40Coefficient.defs.hh File Reference 1750
12.41Coefficient.inlines.hh File Reference 1751
12.42Coefficient.types.hh File Reference 1751
12.43Coefficient_traits_template.hh File Reference 1752
12.44Congruence.cc File Reference 1752
12.45Congruence.defs.hh File Reference 1753
12.46Congruence.inlines.hh File Reference 1754
12.47Congruence.types.hh File Reference 1755
12.48Congruence_System.cc File Reference 1755
12.49Congruence_System.defs.hh File Reference 1756
12.50Congruence_System.inlines.hh File Reference 1757
12.51Congruence_System.types.hh File Reference 1758
12.52Constraint.cc File Reference 1758
12.53Constraint.defs.hh File Reference 1759
12.54Constraint.inlines.hh File Reference 1760
12.55Constraint.types.hh File Reference 1761
12.56Constraint_System.cc File Reference 1761
12.57Constraint_System.defs.hh File Reference 1761
12.58Constraint_System.inlines.hh File Reference 1763
12.59Constraint_System.types.hh File Reference 1763
12.60conversion.cc File Reference 1764

12.61 DB_Matrix.defs.hh File Reference 1764
12.62 DB_Matrix.inlines.hh File Reference 1766
12.63 DB_Matrix.templates.hh File Reference 1767
12.64 DB_Matrix.types.hh File Reference 1768
12.65 DB_Row.defs.hh File Reference 1768
12.66 DB_Row.inlines.hh File Reference 1770
12.67 DB_Row.templates.hh File Reference 1771
12.68 DB_Row.types.hh File Reference 1772
12.69 definitions.dox File Reference 1772
12.70 Determinate.defs.hh File Reference 1772
12.71 Determinate.inlines.hh File Reference 1774
12.72 distances.defs.hh File Reference 1774
12.73 distances.inlines.hh File Reference 1775
12.74 distances.types.hh File Reference 1776
12.75 fdl.dox File Reference 1777
12.76 Float.cc File Reference 1777
12.77 Float.defs.hh File Reference 1777
12.78 Float.inlines.hh File Reference 1779
12.79 Generator.cc File Reference 1779
12.80 Generator.defs.hh File Reference 1780
12.81 Generator.inlines.hh File Reference 1781
12.82 Generator.types.hh File Reference 1782
12.83 Generator_System.cc File Reference 1782
12.84 Generator_System.defs.hh File Reference 1782
12.85 Generator_System.inlines.hh File Reference 1784
12.86 Generator_System.types.hh File Reference 1784
12.87 globals.cc File Reference 1785
12.88 globals.defs.hh File Reference 1785
12.89 globals.inlines.hh File Reference 1792
12.90 globals.types.hh File Reference 1793
12.91 GMP_Integer.defs.hh File Reference 1795
12.92 GMP_Integer.inlines.hh File Reference 1796
12.93 GMP_Integer.types.hh File Reference 1796
12.94 gpl.dox File Reference 1797
12.95 Grid.defs.hh File Reference 1797
12.96 Grid.inlines.hh File Reference 1799

12.97 Grid.templates.hh File Reference 1800
12.98 Grid.types.hh File Reference 1800
12.99 Grid_Certificate.cc File Reference 1801
12.100 Grid_Certificate.defs.hh File Reference 1801
12.101 Grid_Certificate.inlines.hh File Reference 1802
12.102 Grid_Certificate.types.hh File Reference 1803
12.103 Grid_chdims.cc File Reference 1803
12.104 Grid_conversion.cc File Reference 1803
12.105 Grid_Generator.cc File Reference 1804
12.106 Grid_Generator.defs.hh File Reference 1804
12.107 Grid_Generator.inlines.hh File Reference 1805
12.108 Grid_Generator.types.hh File Reference 1806
12.109 Grid_Generator_System.cc File Reference 1806
12.110 Grid_Generator_System.defs.hh File Reference 1806
12.111 Grid_Generator_System.inlines.hh File Reference 1807
12.112 Grid_Generator_System.types.hh File Reference 1808
12.113 Grid_nonpublic.cc File Reference 1809
12.114 Grid_public.cc File Reference 1809
12.115 Grid_simplify.cc File Reference 1810
12.116 Grid_widenings.cc File Reference 1810
12.117 H79_Certificate.cc File Reference 1811
12.118 H79_Certificate.defs.hh File Reference 1811
12.119 H79_Certificate.inlines.hh File Reference 1812
12.120 H79_Certificate.types.hh File Reference 1813
12.121 Init.cc File Reference 1813
12.122 Init.defs.hh File Reference 1815
12.123 Init.inlines.hh File Reference 1816
12.124 Init.types.hh File Reference 1817
12.125 Initializer.hh File Reference 1817
12.126 Interval.defs.hh File Reference 1818
12.127 Interval.inlines.hh File Reference 1820
12.128 Interval.types.hh File Reference 1823
12.129 Interval_Info.defs.hh File Reference 1823
12.130 Interval_Info.inlines.hh File Reference 1825
12.131 Interval_Info.types.hh File Reference 1825
12.132 Interval_Restriction.defs.hh File Reference 1826

12.133Iterator_to_const.defs.hh File Reference 1829
12.134Iterator_to_const.inlines.hh File Reference 1831
12.135Iterator_to_const.types.hh File Reference 1831
12.136Linear_Expression.cc File Reference 1832
12.137Linear_Expression.defs.hh File Reference 1832
12.138Linear_Expression.inlines.hh File Reference 1833
12.139Linear_Expression.types.hh File Reference 1834
12.140Linear_Row.cc File Reference 1835
12.141Linear_Row.defs.hh File Reference 1835
12.142Linear_Row.inlines.hh File Reference 1836
12.143Linear_Row.types.hh File Reference 1837
12.144Linear_System.cc File Reference 1837
12.145Linear_System.defs.hh File Reference 1838
12.146Linear_System.inlines.hh File Reference 1839
12.147Linear_System.types.hh File Reference 1840
12.148math_utilities.defs.hh File Reference 1841
12.149math_utilities.inlines.hh File Reference 1842
12.150Matrix.cc File Reference 1844
12.151Matrix.defs.hh File Reference 1844
12.152Matrix.inlines.hh File Reference 1845
12.153Matrix.types.hh File Reference 1846
12.154meta_programming.hh File Reference 1846
12.155minimize.cc File Reference 1849
12.156MIP_Problem.cc File Reference 1849
12.157MIP_Problem.defs.hh File Reference 1850
12.158MIP_Problem.inlines.hh File Reference 1851
12.159MIP_Problem.templates.hh File Reference 1852
12.160MIP_Problem.types.hh File Reference 1854
12.161namespaces.hh File Reference 1854
12.162NNC_Polyhedron.cc File Reference 1855
12.163NNC_Polyhedron.defs.hh File Reference 1856
12.164NNC_Polyhedron.inlines.hh File Reference 1857
12.165NNC_Polyhedron.types.hh File Reference 1858
12.166Octagonal_Shape.cc File Reference 1858
12.167Octagonal_Shape.defs.hh File Reference 1858
12.168Octagonal_Shape.inlines.hh File Reference 1861

12.169 Octagonal_Shape.templates.hh File Reference 1862
12.170 Octagonal_Shape.types.hh File Reference 1863
12.171 OR_Matrix.defs.hh File Reference 1863
12.172 OR_Matrix.inlines.hh File Reference 1865
12.173 OR_Matrix.templates.hh File Reference 1866
12.174 OR_Matrix.types.hh File Reference 1868
12.175 Partially_Reduced_Product.defs.hh File Reference 1868
12.176 Partially_Reduced_Product.inlines.hh File Reference 1870
12.177 Partially_Reduced_Product.templates.hh File Reference 1871
12.178 Partially_Reduced_Product.types.hh File Reference 1872
12.179 Ph_Status.cc File Reference 1873
12.180 Ph_Status.idefs.hh File Reference 1873
12.181 Ph_Status.inlines.hh File Reference 1873
12.182 PIP_Problem.cc File Reference 1873
12.183 PIP_Problem.defs.hh File Reference 1874
12.184 PIP_Problem.inlines.hh File Reference 1876
12.185 PIP_Problem.templates.hh File Reference 1876
12.186 PIP_Problem.types.hh File Reference 1877
12.187 PIP_Tree.cc File Reference 1877
12.188 PIP_Tree.defs.hh File Reference 1878
12.189 PIP_Tree.inlines.hh File Reference 1880
12.190 PIP_Tree.types.hh File Reference 1880
12.191 Pointset_Powerset.cc File Reference 1880
12.192 Pointset_Powerset.defs.hh File Reference 1881
12.193 Pointset_Powerset.inlines.hh File Reference 1882
12.194 Pointset_Powerset.templates.hh File Reference 1883
12.195 Pointset_Powerset.types.hh File Reference 1885
12.196 Poly_Con_Relation.cc File Reference 1885
12.197 Poly_Con_Relation.defs.hh File Reference 1885
12.198 Poly_Con_Relation.inlines.hh File Reference 1887
12.199 Poly_Con_Relation.types.hh File Reference 1887
12.200 Poly_Gen_Relation.cc File Reference 1887
12.201 Poly_Gen_Relation.defs.hh File Reference 1888
12.202 Poly_Gen_Relation.inlines.hh File Reference 1889
12.203 Poly_Gen_Relation.types.hh File Reference 1889
12.204 Polyhedron.defs.hh File Reference 1890

12.205 Polyhedron.inlines.hh File Reference 1892
12.206 Polyhedron.templates.hh File Reference 1893
12.207 Polyhedron.types.hh File Reference 1894
12.208 Polyhedron_chdims.cc File Reference 1894
12.209 Polyhedron_nonpublic.cc File Reference 1894
12.210 Polyhedron_public.cc File Reference 1895
12.211 Polyhedron_widenings.cc File Reference 1896
12.212 Powerset.defs.hh File Reference 1898
12.213 Powerset.inlines.hh File Reference 1900
12.214 Powerset.templates.hh File Reference 1900
12.215 Powerset.types.hh File Reference 1902
12.216 Row.cc File Reference 1902
12.217 Row.defs.hh File Reference 1903
12.218 Row.inlines.hh File Reference 1904
12.219 Row.types.hh File Reference 1905
12.220 Scalar_Products.cc File Reference 1905
12.221 Scalar_Products.defs.hh File Reference 1905
12.222 Scalar_Products.inlines.hh File Reference 1906
12.223 Scalar_Products.types.hh File Reference 1907
12.224 Simplify.cc File Reference 1907
12.225 stdiobuf.cc File Reference 1908
12.226 stdiobuf.defs.hh File Reference 1908
12.227 stdiobuf.inlines.hh File Reference 1909
12.228 stdiobuf.types.hh File Reference 1909
12.229 Termination.cc File Reference 1910
12.230 Termination.defs.hh File Reference 1911
12.231 Termination.templates.hh File Reference 1914
12.232 Topology.hh File Reference 1917
12.233 Variable.cc File Reference 1918
12.234 Variable.defs.hh File Reference 1918
12.235 Variable.inlines.hh File Reference 1919
12.236 Variable.types.hh File Reference 1920
12.237 Variables_Set.cc File Reference 1920
12.238 Variables_Set.defs.hh File Reference 1921
12.239 Variables_Set.inlines.hh File Reference 1922
12.240 Variables_Set.types.hh File Reference 1923

12.241version.cc File Reference 1923
12.242version.hh File Reference 1924
12.243Widening_Function.defs.hh File Reference 1925
12.244Widening_Function.inlines.hh File Reference 1925
12.245Widening_Function.types.hh File Reference 1926

1 General Information on the PPL

1.1 The Main Features

The Parma Polyhedra Library (PPL) is a modern C++ library for the manipulation of numerical information that can be represented by points in some n-dimensional vector space. For instance, one of the key domains the PPL supports is that of rational convex polyhedra (Section Convex Polyhedra). Such domains are employed in several systems for the analysis and verification of hardware and software components, with applications spanning imperative, functional and logic programming languages, synchronous languages and synchronization protocols, real-time and hybrid systems. Even though the PPL library is not meant to target a particular problem, the design of its interface has been largely influenced by the needs of the above class of applications. That is the reason why the library implements a few operators that are more or less specific to static analysis applications, while lacking some other operators that might be useful when working, e.g., in the field of computational geometry.

The main features of the library are the following:

- it is user friendly: you write `x + 2*y + 5*z <= 7` when you mean it;
- it is fully dynamic: available virtual memory is the only limitation to the dimension of anything;
- it provides full support for the manipulation of convex polyhedra that are not topologically closed;
- it is written in standard C++: meant to be portable;
- it is exception-safe: never leaks resources or leaves invalid object fragments around;
- it is rather efficient: and we hope to make it even more so;
- it is thoroughly documented: perhaps not literate programming but close enough;
- it has interfaces to other programming languages: including C, Java, OCaml and a number of Prolog systems;
- it is free software: distributed under the terms of the GNU General Public License.

In the following section we describe all the domains available to the PPL user. More detailed descriptions of these domains and the operations provided will be found in subsequent sections.

In the final section of this chapter (Section Using the Library), we provide some additional advice on the use of the library.

1.1.1 Semantic Geometric Descriptors

A *semantic geometric descriptor* is a subset of $\mathbb{R}^n$. The PPL provides several classes of semantic GDs. These are identified by their C++ class name, together with the class template parameters, if any. These classes include the *simple classes*:

- C_Polyhedron ,
- NNC_Polyhedron ,
- BD_Shape<T> ,
- Octagonal_Shape<T> ,
- Box<ITV> , and
- Grid ,

where:

- T is a numeric type chosen among mpz_class, mpq_class, signed char, short, int, long, long long (or any of the C99 exact width integer equivalents int8_t, int16_t, and so forth); and
- ITV is an instance of the Interval template class.

Other semantic GDs, the *compound classes*, can be constructed (also recursively) from all the GDs classes. These include:

- Pointset_Powerset<PSET> ,
- Partially_Reduced_Product<D1, D2, R> ,

where PSET, D1 and D2 can be any semantic GD classes and R is the reduction operation to be applied to the component domains of the product class.

A uniform set of operations is provided for creating, testing and maintaining each of the semantic GDs. However, as many of these depend on one or more syntactic GDs, we first describe the syntactic GDs.

1.1.2 Syntactic Geometric Descriptors

A *syntactic geometric descriptor* is for defining, modifying and inspecting a semantic GD. There are three kinds of *syntactic GDs*: *basic GDs*, *constraint GDs* and *generator GDs*. Some of these are *generic* and some *specific*. A generic syntactic GD can be used (in the appropriate context) with any semantic GD; clearly, different semantic GDs will usually provide different levels of support for the different subclasses of generic GDs. In contrast, the use of a specific GD may be restricted to apply to a given subset of the semantic GDs (i.e., some semantic GDs provide no support at all for them).

1.1.2.1 Basic Geometric Descriptors

The following basic GDs currently supported by the PPL are:

- space dimension;
- variable and variable set;
- coefficient;
- linear expression;
- relation symbol;
- vector point.

These classes, which are all generic syntactic GDs, are used to build the constraint and generator GDs as well as support many generic operations on the semantic GDs.

1.1.2.2 Constraint Geometric Descriptors

The PPL currently supports the following classes of *generic* constraint GDs:

- linear constraint;
- linear congruence.

Each linear constraint can be further classified to belong to one or more of the following syntactic subclasses:

- inconsistent constraints (e.g., $0 \geq 2$);
- tautological constraints (e.g., $0 \leq 2$);
- interval constraints (e.g., $x \leq 2$);
- bounded-difference constraints (e.g., $x - y \leq 2$);
- octagonal constraints (e.g., $x + y \leq 2$);
- linear equality constraints (e.g., $x = 2$);
- non-strict linear inequality constraints (e.g., $x - 3y \leq 2$);
- strict linear inequality constraints (e.g., $x - 3y < 2$).

Note that the subclasses are not disjoint.

Similarly, each linear congruence can be classified to belong to one or more of the following syntactic subclasses:

- inconsistent congruences (e.g., $0 \equiv_2 1$);
- tautological congruences (e.g., $0 \equiv_2 2$);
- linear equality, i.e., non-proper congruences (e.g., $x + 3y \equiv_0 0$);
- proper congruences (e.g., $x + 3y \equiv_5 0$).

The library also supports systems, i.e., finite collections, of either linear constraints or linear congruences (but see the note below).

Each semantic GD provides *optimal* support for some of the subclasses of generic syntactic GDs listed above: here, the word "optimal" means that the considered semantic GD computes the *best upward approximation* of the exact meaning of the linear constraint or congruence. When a semantic GD operation is applied to a syntactic GD that is not optimally supported, it will either indicate its unsuitability (e.g., by throwing an exception) or it will apply an upward approximation semantics (possibly not the best one).

For instance, the semantic GD of topologically closed convex polyhedra provides optimal support for non-strict linear inequality and equality constraints, but it does not provide optimal support for strict inequalities. Some of its operations (e.g., `add_constraint` and `add_congruence`) will throw an exception if supplied with a non-trivial strict inequality constraint or a proper congruence; some other operations (e.g., `refine_with_constraint` or `refine_with_congruence`) will compute an over-approximation.

Similarly, the semantic GD of rational boxes (i.e., multi-dimensional intervals) having integral values as interval boundaries provides optimal support for all interval constraints: even though the interval constraint $2x \leq 5$ cannot be represented exactly, it will be optimally approximated by the constraint $x \leq 3$.

Note

When providing an upward approximation for a constraint or congruence, we consider it in isolation: in particular, the approximation of each element of a system of GDs is independent from the other elements; also, the approximation is independent from the current value of the semantic GD.

1.1.2.3 Generator Geometric Descriptors

The PPL currently supports two classes of generator GDs:

- polyhedra generator: these are polyhedra points, rays and lines;
- grid generator: these are grid points, parameters and lines.

Rays, lines and parameters are specific of the mentioned semantic GDs and, therefore, they cannot be used by other semantic GDs. In contrast, as already mentioned above, points are basic geometric descriptors since they are also used in *generic* PPL operations.

1.1.3 Generic Operations on Semantic Geometric Descriptors

1. Constructors of a universe or empty semantic GD with the given space dimension.
2. Operations on a semantic GD that do not depend on the syntactic GDs.
   - `is_empty()`, `is_universe()`, `is_topologically_closed()`, `is_discrete()`, `is_bounded()`, `contains_integer_point()`
     test for the named properties of the semantic GD.
   - `total_memory_in_bytes()`, `external_memory_in_bytes()`
     return the total and external memory size in bytes.
   - `OK()`
     checks that the semantic GD has a valid internal representation. (Some GDs provide this method with an optional Boolean argument that, when true, requires to also check for non-emptiness.)
   - `space_dimension()`, `affine_dimension()`
     return, respectively, the space and affine dimensions of the GD.
   - `add_space_dimensions_and_embed()`, `add_space_dimensions_and_project()`, `expand_space_dimension()`, `remove_space_dimensions()`, `fold_space_dimensions()`, `map_space_dimensions()`
     modify the space dimensions of the semantic GD; where, depending on the operation, the arguments can include the number of space dimensions to be added or removed a variable or set of variables denoting the actual dimensions to be used and a partial function defining a mapping between the dimensions.
   - `contains()`, `strictly_contains()`, `is_disjoint_from()`
     compare the semantic GD with an argument semantic GD of the same class.
   - `topological_closure_assign()`, `intersection_assign()`, `upper_bound_assign()`, `difference_assign()`, `time_elapse_assign()`, `widening_assign()`, `concatenate_assign()`, `swap()`
     modify the semantic GD, possibly with an argument semantic GD of the same class.
   - `constrains()`, `bounds_from_above()`, `bounds_from_below()`, `maximize()`, `minimize()`.
     These find information about the bounds of the semantic GD where the argument variable or linear expression define the direction of the bound.

- `affine_image()`, `affine_preimage()`, `generalized_affine_image()`, `generalized_affine_preimage()`, `bounded_affine_image()`, `bounded_affine_preimage()`.

  These perform several variations of the affine image and preimage operations where, depending on the operation, the arguments can include a variable representing the space dimension to which the transformation will be applied and linear expressions with possibly a relation symbol and denominator value that define the exact form of the transformation.

- `ascii_load()`, `ascii_dump()`

  are the ascii input and output operations.

3. Constructors of a semantic GD of one class from a semantic GD of any other class. These constructors obey an *upward approximation semantics*, meaning that the constructed semantic GD is guaranteed to contain all the points of the source semantic GD, but possibly more. Some of these constructors provide a complexity parameter with which the application can control the complexity/precision trade-off for the construction operation: by using the complexity parameter, it is possible to keep the construction operation in the polynomial or the simplex worst-case complexity class, possibly incurring into a further upward approximation if the precise constructor is based on an algorithm having exponential complexity.

4. Constructors of a semantic GD from a constraint GD; either a linear constraint system or a linear congruence system. These constructors assume that the given semantic GD provides optimal support for the argument syntactic GD: if that is not the case, an invalid argument exception is thrown.

5. Other interaction between the semantic GDs and constraint GDs.

   - `add_constraint()`, `add_constraints()`, `add_recycled_constraints()`, `add_congruence()`, `add_congruences()`, `add_recycled_congruences()`.

     These methods assume that the given semantic GD provides optimal support for the argument syntactic GD: if that is not the case, an invalid argument exception is thrown.

     For `add_recycled_constraints()` and `add_recycled_congruences()`, the only assumption that can be made on the constraint GD after return (successful or exceptional) is that it can be safely destroyed.

   - `refine_with_constraint()`, `refine_with_constraints()`, `refine_with_congruence()`, `refine_with_congruences()`.

     If the argument constraint GD is optimally supported by the semantic GD, the methods behave the same as the corresponding `add_*` methods listed above. Otherwise the constraint GD is used only to a limited extent to refine the semantic GD; possibly not at all. Notice that, while repeating an add operation is pointless, this is not true for the refine operations. For example, in those cases where

     ```
     Semantic_GD.add_constraint(c)
     ```

     raises an exception, a fragment of the form

     ```
     Semantic_GD.refine_with_constraint(c)
     // Other add_constraint(s) or refine_with_constraint(s) operations
     // on Semantic_GD.
     Semantic_GD.refine_with_constraint(c)
     ```

     may give more precise results than a single

     ```
     Semantic_GD.refine_with_constraint(c).
     // Other add_constraint(s) or refine_with_constraint(s) operations
     // on Semantic_GD.
     ```

   - `constraints()`, `minimized_constraints()`, `congruences()`, `minimized_congruences()`.

     Returns the indicated system of constraint GDs satisfied by the semantic GD.

- `can_recycle_constraint_systems()`, `can_recycle_congruence_systems()`.
  Return true if and only if the semantic GD can recycle the indicated constraint GD.
- `relation_with()`.
  This takes a constraint GD as an argument and returns the relations holding between the semantic GD and the constraint GD. The possible relations are: `IS_INCLUDED()`, `SATURATES()`, `STRICTLY_INTERSECTS()`, `IS_DISJOINT()` and `NOTHING()`. This operator also can take a polyhedron generator GD as an argument and returns the relation `SUBSUMES()` or `NOTHING()` that holds between the generator GD and the semantic GD.

1.2 Upward Approximation

The Parma Polyhedra Library, for those cases where an exact result cannot be computed within the specified complexity limits, computes an *upward approximation* of the exact result. For semantic GDs this means that the computed result is a possibly strict superset of the set of points of $\mathbb{R}^n$ that constitutes the exact result. Notice that the PPL does not provide direct support to compute *downward approximations* (i.e., possibly strict subsets of the exact results). While downward approximations can often be computed from upward ones, the required algorithms and the conditions upon which they are correct are outside the current scope of the PPL. Beware, in particular, of the following possible pitfall: the library provides methods to compute upward approximations of set-theoretic difference, which is antitone in its second argument. Applying a difference method to a second argument that is not an exact representation or a downward approximation of reality, would yield a result that, of course, is not an upward approximation of reality. It is the responsibility of the library user to provide the PPL's method with approximations of reality that are consistent with respect to the desired results.

1.3 Approximating Integers

The Parma Polyhedra Library provides support for approximating integer computations using the geometric descriptors it provides. In this section we briefly explain these facilities.

1.3.1 Dropping Non-Integer Points

When a geometric descriptor is used to approximate integer quantities, all the points with non-integral coordinates represent an imprecision of the description. Of course, removing all these points may be impossible (because of convexity) or too expensive. The PPL provides the operator `drop_some_non_integer_points` to possibly tighten a descriptor by dropping some points with non-integer coordinates, using algorithms whose complexity is bounded by a parameter. The set of dimensions that represent integer quantities can be optionally specified. It is worth to stress the role of *some* in the operator name: in general no optimality guarantee is provided.

1.3.2 Approximating Bounded Integers

The Parma Polyhedra Library provides services that allow to compute correct approximations of bounded arithmetic as available in widespread programming languages. Supported bit-widths are 8, 16, 32 and 64 bits, with some limited support for 128 bits. Supported representations are binary unsigned and two's complement signed. Supported overflow behaviors are:

Wrapping: this means that, for a w-bit bounded integer, the computation happens modulo 2^w. In turn, this signifies that the computation happens *as if* the unbounded arithmetic result was computed and

then wrapped. For unsigned integers, the wrapping function is simply $x \bmod 2^w$, most conveniently defined as

$$\mathrm{wrap}^{\mathrm{u}}_w(x) \stackrel{\mathrm{def}}{=} x - 2^w \lfloor x/2^w \rfloor.$$

For signed integers the wrapping function is, instead,

$$\mathrm{wrap}^{\mathrm{s}}_w(x) \stackrel{\mathrm{def}}{=} \begin{cases} \mathrm{wrap}^{\mathrm{u}}_w(x), & \text{if } \mathrm{wrap}^{\mathrm{u}}_w(x) < 2^{w-1}; \\ \mathrm{wrap}^{\mathrm{u}}_w(x) - 2^w, & \text{otherwise.} \end{cases}$$

Undefined: this means that the result of the operation resulting in an overflow can take any value. This is useful to partially model systems where overflow has unspecified effects on the computed result. Even though something more serious can happen in the system being analyzed ---due to, e.g., C's undefined behavior---, here we are only concerned with the results of arithmetic operations. It is the responsibility of the analyzer to ensure that other manifestations of undefined behavior are conservatively approximated.

Impossible: this is for the analysis of languages where overflow is trapped before it affects the state, for which, thus, any indication that an overflow may have affected the state is necessarily due to the imprecision of the analysis.

1.3.2.1 Wrapping Operator

One possibility for precisely approximating the semantics of programs that operate on bounded integer variables is to follow the approach described in [SK07]. The idea is to associate space dimensions to the *unwrapped values* of bounded variables. Suppose $\mathtt{j}$ is a w-bit, unsigned program variable associated to a space dimension labeled by the variable x. If x is constrained by some numerical abstraction to take values in a set $S \subseteq \mathbb{R}$, then the program variable $\mathtt{j}$ can only take values in $\left\{ \mathrm{wrap}^{\mathrm{u}}_w(z) \;\middle|\; z \in S \right\}$. There are two reasons why this is interesting: firstly, this allows for the retention of relational information by using a single numerical abstraction tracking multiple program variables. Secondly, the integers modulo 2^w form a ring of equivalence classes on which addition and multiplication are well defined. This means, e.g., that assignments with affine right-hand sides and involving only variables with the same bit-width and representation can be safely modeled by affine images. While upper bounds and widening can be used without any precaution, anything that can be reconducted to intersection requires a preliminary *wrapping* phase, where the dimensions corresponding to bounded integer types are brought back to their natural domain. This necessity arises naturally for the analysis of conditionals and conversion operators, as well as in the realization of domain combinations.

The PPL provides a general wrapping operator that is parametric with respect to the set of space dimensions (variables) to be wrapped, the width, representation and overflow behavior of all these variables. An optional constraint system can, when given, improve the precision. This constraint system, which must only depend on variables with respect to which wrapping is performed, is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation afterwards. The general wrapping operator offered by the PPL also allows control of the complexity/precision ratio by means of two additional parameters: an unsigned integer encoding a complexity threshold, with higher values resulting in possibly improved precision; and a Boolean controlling whether space dimensions should be wrapped individually, something that results in much greater efficiency to the detriment of precision, or collectively.

Note that the PPL assumes that any space dimension subject to wrapping is being used to capture the value of bounded integer values. As a consequence the library is free to drop, from the involved numerical abstraction, any point having a non-integer coordinate that corresponds to a space dimension subject to wrapping. It must be stressed that freedom to drop such points does not constitute an obligation to remove

all of them (especially because this would be extraordinarily expensive on some numerical abstractions). The PPL provides operators for the more systematic removal of points with non-integral coordinates.

The wrapping operator will only remove some of these points as a by-product of its main task and only when this comes at a negligible extra cost.

1.4 Convex Polyhedra

In this section we introduce convex polyhedra, as considered by the library, in more detail. For more information about the definitions and results stated here see [BRZH02b], [Fuk98], [NW88], and [Wil93].

1.4.1 Vectors, Matrices and Scalar Products

We denote by $\mathbb{R}^n$ the n-dimensional vector space on the field of real numbers $\mathbb{R}$, endowed with the standard topology. The set of all non-negative reals is denoted by $\mathbb{R}_+$. For each $i \in \{0, \ldots, n-1\}$, v_i denotes the i-th component of the (column) vector $\boldsymbol{v} = (v_0, \ldots, v_{n-1})^{\mathrm{T}} \in \mathbb{R}^n$. We denote by $\mathbf{0}$ the vector of $\mathbb{R}^n$, called *the origin*, having all components equal to zero. A vector $\boldsymbol{v} \in \mathbb{R}^n$ can be also interpreted as a matrix in $\mathbb{R}^{n \times 1}$ and manipulated accordingly using the usual definitions for addition, multiplication (both by a scalar and by another matrix), and transposition, denoted by $\boldsymbol{v}^{\mathrm{T}}$.

The *scalar product* of $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$, denoted $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$, is the real number

$$\boldsymbol{v}^{\mathrm{T}} \boldsymbol{w} = \sum_{i=0}^{n-1} v_i w_i.$$

For any $S_1, S_2 \subseteq \mathbb{R}^n$, the *Minkowski's sum* of S_1 and S_2 is: $S_1 + S_2 = \{ \boldsymbol{v}_1 + \boldsymbol{v}_2 \mid \boldsymbol{v}_1 \in S_1, \boldsymbol{v}_2 \in S_2 \}$.

1.4.2 Affine Hyperplanes and Half-spaces

For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, where $\boldsymbol{a} \neq \mathbf{0}$, and for each relation symbol $\bowtie \in \{=, \geq, >\}$, the linear constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ defines:

- an affine hyperplane if it is an equality constraint, i.e., if $\bowtie \in \{=\}$;
- a topologically closed affine half-space if it is a non-strict inequality constraint, i.e., if $\bowtie \in \{\geq\}$;
- a topologically open affine half-space if it is a strict inequality constraint, i.e., if $\bowtie \in \{>\}$.

Note that each hyperplane $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ can be defined as the intersection of the two closed affine half-spaces $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ and $\langle -\boldsymbol{a}, \boldsymbol{x} \rangle \geq -b$. Also note that, when $\boldsymbol{a} = \mathbf{0}$, the constraint $\langle \mathbf{0}, \boldsymbol{x} \rangle \bowtie b$ is either a tautology (i.e., always true) or inconsistent (i.e., always false), so that it defines either the whole vector space $\mathbb{R}^n$ or the empty set $\varnothing$.

1.4.3 Convex Polyhedra

The set $\mathcal{P} \subseteq \mathbb{R}^n$ is a *not necessarily closed convex polyhedron* (*NNC polyhedron*, for short) if and only if either $\mathcal{P}$ can be expressed as the intersection of a finite number of (open or closed) affine half-spaces of $\mathbb{R}^n$ or $n = 0$ and $\mathcal{P} = \varnothing$. The set of all NNC polyhedra on the vector space $\mathbb{R}^n$ is denoted $\mathbb{P}_n$.

The set $\mathcal{P} \in \mathbb{P}_n$ is a *closed convex polyhedron* (*closed polyhedron*, for short) if and only if either $\mathcal{P}$ can be expressed as the intersection of a finite number of closed affine half-spaces of $\mathbb{R}^n$ or $n = 0$ and $\mathcal{P} = \varnothing$. The set of all closed polyhedra on the vector space $\mathbb{R}^n$ is denoted $\mathbb{CP}_n$.

When ordering NNC polyhedra by the set inclusion relation, the empty set $\varnothing$ and the vector space $\mathbb{R}^n$ are, respectively, the smallest and the biggest elements of both $\mathbb{P}_n$ and $\mathbb{CP}_n$. The vector space $\mathbb{R}^n$ is also called the *universe* polyhedron.

In theoretical terms, $\mathbb{P}_n$ is a *lattice* under set inclusion and $\mathbb{CP}_n$ is a *sub-lattice* of $\mathbb{P}_n$.

Note

In the following, we will usually specify operators on the domain $\mathbb{P}_n$ of NNC polyhedra. Unless an explicit distinction is made, these operators are provided with the same specification when applied to the domain $\mathbb{CP}_n$ of topologically closed polyhedra. The implementation maintains a clearer separation between the two domains of polyhedra (see Topologies and Topological-compatibility): while computing polyhedra in $\mathbb{P}_n$ may provide more precise results, polyhedra in $\mathbb{CP}_n$ can be represented and manipulated more efficiently. As a rule of thumb, if your application will only manipulate polyhedra that are topologically closed, then it should use the simpler domain $\mathbb{CP}_n$. Using NNC polyhedra is only recommended if you are going to actually benefit from the increased accuracy.

1.4.4 Bounded Polyhedra

An NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is *bounded* if there exists a $\lambda \in \mathbb{R}_+$ such that:

$$\mathcal{P} \subseteq \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; -\lambda \leq x_j \leq \lambda \text{ for } j = 0, \ldots, n-1 \right\}.$$

A bounded polyhedron is also called a *polytope*.

1.5 Representations of Convex Polyhedra

NNC polyhedra can be specified by using two possible representations, the constraints (or implicit) representation and the generators (or parametric) representation.

1.5.1 Constraints Representation

In the sequel, we will simply write "equality" and "inequality" to mean "linear equality" and "linear inequality", respectively; also, we will refer to either an equality or an inequality as a *constraint*.

By definition, each polyhedron $\mathcal{P} \in \mathbb{P}_n$ is the set of solutions to a *constraint system*, i.e., a finite number of constraints. By using matrix notation, we have

$$\mathcal{P} \stackrel{\text{def}}{=} \{ \boldsymbol{x} \in \mathbb{R}^n \mid A_1 \boldsymbol{x} = \boldsymbol{b}_1, A_2 \boldsymbol{x} \geq \boldsymbol{b}_2, A_3 \boldsymbol{x} > \boldsymbol{b}_3 \},$$

where, for all $i \in \{1, 2, 3\}$, $A_i \in \mathbb{R}^{m_i} \times \mathbb{R}^n$ and $\boldsymbol{b}_i \in \mathbb{R}^{m_i}$, and $m_1, m_2, m_3 \in \mathbb{N}$ are the number of equalities, the number of non-strict inequalities, and the number of strict inequalities, respectively.

1.5.2 Combinations and Hulls

Let $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} \subseteq \mathbb{R}^n$ be a finite set of vectors. For all scalars $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the vector $\boldsymbol{v} = \sum_{j=1}^{k} \lambda_j \boldsymbol{x}_j$ is said to be a *linear* combination of the vectors in S. Such a combination is said to be

- a *positive* (or *conic*) combination, if $\forall j \in \{1, \ldots, k\} : \lambda_j \in \mathbb{R}_+$;
- an *affine* combination, if $\sum_{j=1}^{k} \lambda_j = 1$;
- a *convex* combination, if it is both positive and affine.

We denote by $\mathrm{linear.hull}(S)$ (resp., $\mathrm{conic.hull}(S)$, $\mathrm{affine.hull}(S)$, $\mathrm{convex.hull}(S)$) the set of all the linear (resp., positive, affine, convex) combinations of the vectors in S.

Let $P, C \subseteq \mathbb{R}^n$, where $P \cup C = S$. We denote by $\mathrm{nnc.hull}(P, C)$ the set of all convex combinations of the vectors in S such that $\lambda_j > 0$ for some $\boldsymbol{x}_j \in P$ (informally, we say that there exists a vector of P that plays an active role in the convex combination). Note that $\mathrm{nnc.hull}(P, C) = \mathrm{nnc.hull}(P, P \cup C)$ so that, if $C \subseteq P$,

$$\mathrm{convex.hull}(P) = \mathrm{nnc.hull}(P, \varnothing) = \mathrm{nnc.hull}(P, P) = \mathrm{nnc.hull}(P, C).$$

It can be observed that $\mathrm{linear.hull}(S)$ is an affine space, $\mathrm{conic.hull}(S)$ is a topologically closed convex cone, $\mathrm{convex.hull}(S)$ is a topologically closed polytope, and $\mathrm{nnc.hull}(P, C)$ is an NNC polytope.

1.5.3 Points, Closure Points, Rays and Lines

Let $\mathcal{P} \in \mathbb{P}_n$ be an NNC polyhedron. Then

- a vector $\boldsymbol{p} \in \mathcal{P}$ is called a *point* of $\mathcal{P}$;
- a vector $\boldsymbol{c} \in \mathbb{R}^n$ is called a *closure point* of $\mathcal{P}$ if it is a point of the topological closure of $\mathcal{P}$;
- a vector $\boldsymbol{r} \in \mathbb{R}^n$, where $\boldsymbol{r} \neq \boldsymbol{0}$, is called a *ray* (or direction of infinity) of $\mathcal{P}$ if $\mathcal{P} \neq \varnothing$ and $\boldsymbol{p} + \lambda \boldsymbol{r} \in \mathcal{P}$, for all points $\boldsymbol{p} \in \mathcal{P}$ and all $\lambda \in \mathbb{R}_+$;
- a vector $\boldsymbol{l} \in \mathbb{R}^n$ is called a *line* of $\mathcal{P}$ if both $\boldsymbol{l}$ and $-\boldsymbol{l}$ are rays of $\mathcal{P}$.

A point of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is a *vertex* if and only if it cannot be expressed as a convex combination of any other pair of distinct points in $\mathcal{P}$. A ray $\boldsymbol{r}$ of a polyhedron $\mathcal{P}$ is an *extreme ray* if and only if it cannot be expressed as a positive combination of any other pair $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ of rays of $\mathcal{P}$, where $\boldsymbol{r} \neq \lambda \boldsymbol{r}_1$, $\boldsymbol{r} \neq \lambda \boldsymbol{r}_2$ and $\boldsymbol{r}_1 \neq \lambda \boldsymbol{r}_2$ for all $\lambda \in \mathbb{R}_+$ (i.e., rays differing by a positive scalar factor are considered to be the same ray).

1.5.4 Generators Representation

Each NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ can be represented by finite sets of lines L, rays R, points P and closure points C of $\mathcal{P}$. The 4-tuple $\mathcal{G} = (L, R, P, C)$ is said to be a *generator system* for $\mathcal{P}$, in the sense that

$$\mathcal{P} = \mathrm{linear.hull}(L) + \mathrm{conic.hull}(R) + \mathrm{nnc.hull}(P, C),$$

where the symbol '$+$' denotes the Minkowski's sum.

When $\mathcal{P} \in \mathbb{CP}_n$ is a closed polyhedron, then it can be represented by finite sets of lines L, rays R and points P of $\mathcal{P}$. In this case, the 3-tuple $\mathcal{G} = (L, R, P)$ is said to be a *generator system* for $\mathcal{P}$ since we have

$$\mathcal{P} = \mathrm{linear.hull}(L) + \mathrm{conic.hull}(R) + \mathrm{convex.hull}(P).$$

Thus, in this case, every closure point of $\mathcal{P}$ is a point of $\mathcal{P}$.

For any $\mathcal{P} \in \mathbb{P}_n$ and generator system $\mathcal{G} = (L, R, P, C)$ for $\mathcal{P}$, we have $\mathcal{P} = \varnothing$ if and only if $P = \varnothing$. Also P must contain all the vertices of $\mathcal{P}$ although $\mathcal{P}$ can be non-empty and have no vertices. In this case, as P is necessarily non-empty, it must contain points of $\mathcal{P}$ that are *not* vertices. For instance, the half-space of $\mathbb{R}^2$ corresponding to the single constraint $y \geq 0$ can be represented by the generator system $\mathcal{G} = (L, R, P, C)$ such that $L = \{(1, 0)^{\mathrm{T}}\}$, $R = \{(0, 1)^{\mathrm{T}}\}$, $P = \{(0, 0)^{\mathrm{T}}\}$, and $C = \varnothing$. It is also worth noting that the only ray in R is *not* an extreme ray of $\mathcal{P}$.

1.5.5 Minimized Representations

A constraints system $\mathcal{C}$ for an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is said to be *minimized* if no proper subset of $\mathcal{C}$ is a constraint system for $\mathcal{P}$.

Similarly, a generator system $\mathcal{G} = (L, R, P, C)$ for an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is said to be *minimized* if there does not exist a generator system $\mathcal{G}' = (L', R', P', C') \neq \mathcal{G}$ for $\mathcal{P}$ such that $L' \subseteq L$, $R' \subseteq R$, $P' \subseteq P$ and $C' \subseteq C$.

1.5.6 Double Description

Any NNC polyhedron $\mathcal{P}$ can be described by using a constraint system $\mathcal{C}$, a generator system $\mathcal{G}$, or both by means of the *double description pair (DD pair)* $(\mathcal{C}, \mathcal{G})$. The *double description method* is a collection of well-known as well as novel theoretical results showing that, given one kind of representation, there are algorithms for computing a representation of the other kind and for minimizing both representations by removing redundant constraints/generators.

Such changes of representation form a key step in the implementation of many operators on NNC polyhedra: this is because some operators, such as intersections and poly-hulls, are provided with a natural and efficient implementation when using one of the representations in a DD pair, while being rather cumbersome when using the other.

1.5.7 Topologies and Topological-compatibility

As indicated above, when an NNC polyhedron $\mathcal{P}$ is necessarily closed, we can ignore the closure points contained in its generator system $\mathcal{G} = (L, R, P, C)$ (as every closure point is also a point) and represent $\mathcal{P}$ by the triple (L, R, P). Similarly, $\mathcal{P}$ can be represented by a constraint system that has no strict inequalities. Thus a necessarily closed polyhedron can have a smaller representation than one that is not necessarily closed. Moreover, operators restricted to work on closed polyhedra only can be implemented more efficiently. For this reason the library provides two alternative "topological kinds" for a polyhedron, *NNC* and *C*. We shall abuse terminology by referring to the topological kind of a polyhedron as its *topology*.

In the library, the topology of each polyhedron object is fixed once for all at the time of its creation and must be respected when performing operations on the polyhedron.

Unless it is otherwise stated, all the polyhedra, constraints and/or generators in any library operation must obey the following *topological-compatibility* rules:

- polyhedra are topologically-compatible if and only if they have the same topology;
- all constraints except for strict inequality constraints and all generators except for closure points are topologically-compatible with both C and NNC polyhedra;
- strict inequality constraints and closure points are topologically-compatible with a polyhedron if and only if it is NNC.

Wherever possible, the library provides methods that, starting from a polyhedron of a given topology, build the corresponding polyhedron having the other topology.

1.5.8 Space Dimensions and Dimension Compatibility

The *space dimension* of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ (resp., a C polyhedron $\mathcal{P} \in \mathbb{CP}_n$) is the dimension $n \in \mathbb{N}$ of the corresponding vector space $\mathbb{R}^n$. The space dimension of constraints, generators and other objects of the library is defined similarly.

Unless it is otherwise stated, all the polyhedra, constraints and/or generators in any library operation must obey the following (space) *dimension-compatibility* rules:

- polyhedra are dimension-compatible if and only if they have the same space dimension;
- the constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ where $\bowtie \in \{=, \geq, >\}$ and $\boldsymbol{a}, \boldsymbol{x} \in \mathbb{R}^m$, is dimension-compatible with a polyhedron having space dimension n if and only if $m \leq n$;
- the generator $\boldsymbol{x} \in \mathbb{R}^m$ is dimension-compatible with a polyhedron having space dimension n if and only if $m \leq n$;
- a system of constraints (resp., generators) is dimension-compatible with a polyhedron if and only if all the constraints (resp., generators) in the system are dimension-compatible with the polyhedron.

While the space dimension of a constraint, a generator or a system thereof is automatically adjusted when needed, the space dimension of a polyhedron can only be changed by explicit calls to operators provided for that purpose.

1.5.9 Affine Independence and Affine Dimension

A finite set of points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} \subseteq \mathbb{R}^n$ is *affinely independent* if, for all $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the system of equations

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i = \boldsymbol{0}, \quad \sum_{i=1}^{k} \lambda_i = 0$$

implies that, for each $i = 1, \ldots, k$, $\lambda_i = 0$.

The maximum number of affinely independent points in $\mathbb{R}^n$ is $n + 1$.

A *non-empty* NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ has *affine dimension* $k \in \mathbb{N}$, denoted by $\dim(\mathcal{P}) = k$, if the maximum number of affinely independent points in $\mathcal{P}$ is $k + 1$.

We remark that the above definition only applies to polyhedra that are not empty, so that $0 \leq \dim(\mathcal{P}) \leq n$. By convention, the affine dimension of an empty polyhedron is 0 (even though the "natural" generalization of the definition above would imply that the affine dimension of an empty polyhedron is -1).

Note

> The affine dimension $k \leq n$ of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ must not be confused with the space dimension n of $\mathcal{P}$, which is the dimension of the enclosing vector space $\mathbb{R}^n$. In particular, we can have $\dim(\mathcal{P}) \neq \dim(\mathcal{Q})$ even though $\mathcal{P}$ and $\mathcal{Q}$ are dimension-compatible; and vice versa, $\mathcal{P}$ and $\mathcal{Q}$ may be dimension-incompatible polyhedra even though $\dim(\mathcal{P}) = \dim(\mathcal{Q})$.

1.5.10 Rational Polyhedra

An NNC polyhedron is called *rational* if it can be represented by a constraint system where all the constraints have rational coefficients. It has been shown that an NNC polyhedron is rational if and only if it can be represented by a generator system where all the generators have rational coefficients.

The library only supports rational polyhedra. The restriction to rational numbers applies not only to polyhedra, but also to the other numeric arguments that may be required by the operators considered, such as the coefficients defining (rational) affine transformations.

1.6 Operations on Convex Polyhedra

In this section we briefly describe operations on NNC polyhedra that are provided by the library.

1.6.1 Intersection and Convex Polyhedral Hull

For any pair of NNC polyhedra $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$, the *intersection* of $\mathcal{P}_1$ and $\mathcal{P}_2$, defined as the set intersection $\mathcal{P}_1 \cap \mathcal{P}_2$, is the biggest NNC polyhedron included in both $\mathcal{P}_1$ and $\mathcal{P}_2$; similarly, the *convex polyhedral hull* (or *poly-hull*) of $\mathcal{P}_1$ and $\mathcal{P}_2$, denoted by $\mathcal{P}_1 \uplus \mathcal{P}_2$, is the smallest NNC polyhedron that includes both $\mathcal{P}_1$ and $\mathcal{P}_2$. The intersection and poly-hull of any pair of closed polyhedra in $\mathbb{CP}_n$ is also closed.

In theoretical terms, the intersection and poly-hull operators defined above are the binary *meet* and the binary *join* operators on the lattices $\mathbb{P}_n$ and $\mathbb{CP}_n$.

1.6.2 Convex Polyhedral Difference

For any pair of NNC polyhedra $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$, the *convex polyhedral difference* (or *poly-difference*) of $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined as the smallest convex polyhedron containing the set-theoretic difference of $\mathcal{P}_1$ and $\mathcal{P}_2$.

In general, even though $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{CP}_n$ are topologically closed polyhedra, their poly-difference may be a convex polyhedron that is not topologically closed. For this reason, when computing the poly-difference of two C polyhedra, the library will enforce the topological closure of the result.

1.6.3 Concatenating Polyhedra

Viewing a polyhedron as a set of tuples (its points), it is sometimes useful to consider the set of tuples obtained by concatenating an ordered pair of polyhedra. Formally, the *concatenation* of the polyhedra $\mathcal{P} \in \mathbb{P}_n$ and $\mathcal{Q} \in \mathbb{P}_m$ (taken in this order) is the polyhedron $\mathcal{R} \in \mathbb{P}_{n+m}$ such that

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ (x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1})^{\mathrm{T}} \in \mathbb{R}^{n+m} \;\middle|\; (x_0, \ldots, x_{n-1})^{\mathrm{T}} \in \mathcal{P}, (y_0, \ldots, y_{m-1})^{\mathrm{T}} \in \mathcal{Q} \right\}.$$

Another way of seeing it is as follows: first embed polyhedron $\mathcal{P}$ into a vector space of dimension $n + m$ and then add a suitably renamed-apart version of the constraints defining $\mathcal{Q}$.

1.6.4 Adding New Dimensions to the Vector Space

The library provides two operators for adding a number i of space dimensions to an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$, therefore transforming it into a new NNC polyhedron $\mathcal{Q} \in \mathbb{P}_{n+i}$. In both cases, the added dimensions of the vector space are those having the highest indices.

The operator `add_space_dimensions_and_embed` *embeds* the polyhedron $\mathcal{P}$ into the new vector space of dimension $i + n$ and returns the polyhedron $\mathcal{Q}$ defined by all and only the constraints defining $\mathcal{P}$ (the variables corresponding to the added dimensions are unconstrained). For instance, when starting from a polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the polyhedron

$$\mathcal{Q} = \left\{ (x_0, x_1, x_2)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x_0, x_1)^{\mathrm{T}} \in \mathcal{P} \right\}.$$

In contrast, the operator `add_space_dimensions_and_project` *projects* the polyhedron $\mathcal{P}$ into the new vector space of dimension $i + n$ and returns the polyhedron $\mathcal{Q}$ whose constraint system, besides the constraints defining $\mathcal{P}$, will include additional constraints on the added dimensions. Namely, the corresponding variables are all constrained to be equal to 0. For instance, when starting from a polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the polyhedron

$$\mathcal{Q} = \left\{ (x_0, x_1, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x_0, x_1)^{\mathrm{T}} \in \mathcal{P} \right\}.$$

1.6.5 Removing Dimensions from the Vector Space

The library provides two operators for removing space dimensions from an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$, therefore transforming it into a new NNC polyhedron $\mathcal{Q} \in \mathbb{P}_m$ where $m \leq n$.

Given a set of variables, the operator `remove_space_dimensions` removes all the space dimensions specified by the variables in the set. For instance, letting $\mathcal{P} \in \mathbb{P}_4$ be the singleton set $\{(3,1,0,2)^{\mathrm{T}}\} \subseteq \mathbb{R}^4$, then after invoking this operator with the set of variables $\{x_1, x_2\}$ the resulting polyhedron is

$$\mathcal{Q} = \{(3,2)^{\mathrm{T}}\} \subseteq \mathbb{R}^2.$$

Given a space dimension m less than or equal to that of the polyhedron, the operator `remove_higher_space_dimensions` removes the space dimensions having indices greater than or equal to m. For instance, letting $\mathcal{P} \in \mathbb{P}_4$ defined as before, by invoking this operator with $m = 2$ the resulting polyhedron will be

$$\mathcal{Q} = \{(3,1)^{\mathrm{T}}\} \subseteq \mathbb{R}^2.$$

1.6.6 Mapping the Dimensions of the Vector Space

The operator `map_space_dimensions` provided by the library maps the dimensions of the vector space $\mathbb{R}^n$ according to a partial injective function $\rho\colon \{0,\ldots,n-1\} \rightarrowtail \mathbb{N}$ such that $\rho\big(\{0,\ldots,n-1\}\big) = \{0,\ldots,m-1\}$ with $m \leq n$. Dimensions corresponding to indices that are not mapped by ρ are removed.

If $m = 0$, i.e., if the function ρ is undefined everywhere, then the operator projects the argument polyhedron $\mathcal{P} \in \mathbb{P}_n$ onto the zero-dimension space $\mathbb{R}^0$; otherwise the result is $\mathcal{Q} \in \mathbb{P}_m$ given by

$$\mathcal{Q} \stackrel{\text{def}}{=} \left\{ \left(v_{\rho^{-1}(0)}, \ldots, v_{\rho^{-1}(m-1)}\right)^{\mathrm{T}} \;\middle|\; (v_0,\ldots,v_{n-1})^{\mathrm{T}} \in \mathcal{P} \right\}.$$

1.6.7 Expanding One Dimension of the Vector Space to Multiple Dimensions

The operator `expand_space_dimension` provided by the library adds m new space dimensions to a polyhedron $\mathcal{P} \in \mathbb{P}_n$, with $n > 0$, so that dimensions n, $n+1$, …, $n+m-1$ of the result $\mathcal{Q}$ are exact copies of the i-th space dimension of $\mathcal{P}$. More formally,

$$\mathcal{Q} \stackrel{\text{def}}{=} \left\{ \boldsymbol{u} \in \mathbb{R}^{n+m} \;\middle|\; \begin{array}{l} \exists \boldsymbol{v}, \boldsymbol{w} \in \mathcal{P} \,.\, u_i = v_i \\ \qquad \wedge \forall j = n, n+1, \ldots, n+m-1 : u_j = w_i \\ \qquad \wedge \forall k = 0, \ldots, n-1 : k \neq i \implies u_k = v_k = w_k \end{array} \right\}.$$

This operation has been proposed in [GDDetal04].

1.6.8 Folding Multiple Dimensions of the Vector Space into One Dimension

The operator `fold_space_dimensions` provided by the library, given a polyhedron $\mathcal{P} \in \mathbb{P}_n$, with $n > 0$, folds a set of space dimensions $J = \{j_0, \ldots, j_{m-1}\}$, with $m < n$ and $j < n$ for each $j \in J$, into space dimension $i < n$, where $i \notin J$. The result is given by

$$\mathcal{Q} \stackrel{\text{def}}{=} \biguplus_{d=0}^{m} \mathcal{Q}_d$$

where

$$\mathcal{Q}_m \stackrel{\text{def}}{=} \left\{ \boldsymbol{u} \in \mathbb{R}^{n-m} \;\middle|\; \begin{array}{l} \exists \boldsymbol{v} \in \mathcal{P} \,.\, u_{i'} = v_i \\ \qquad \wedge \forall k = 0, \ldots, n-1 : k \neq i \implies u_{k'} = v_k \end{array} \right\}$$

and, for $d = 0, \ldots, m-1$,

$$\mathcal{Q}_d \stackrel{\text{def}}{=} \left\{ \boldsymbol{u} \in \mathbb{R}^{n-m} \;\middle|\; \begin{array}{l} \exists \boldsymbol{v} \in \mathcal{P} \,.\, u_{i'} = v_{j_d} \\ \qquad \wedge \forall k = 0, \ldots, n-1 : k \neq i \implies u_{k'} = v_k \end{array} \right\},$$

and, finally, for $k = 0, \ldots, n-1$,

$$k' \stackrel{\text{def}}{=} k - \#\{\, j \in J \mid k > j \,\},$$

($\# S$ denotes the cardinality of the finite set S).

This operation has been proposed in [GDDetal04].

1.6.9 Images and Preimages of Affine Transfer Relations

For each relation $\phi \subseteq \mathbb{R}^n \times \mathbb{R}^m$, we denote by $\phi(S) \subseteq \mathbb{R}^m$ the *image* under ϕ of the set $S \subseteq \mathbb{R}^n$; formally,

$$\phi(S) \stackrel{\text{def}}{=} \left\{\, \boldsymbol{w} \in \mathbb{R}^m \;\middle|\; \exists \boldsymbol{v} \in S \,.\, (\boldsymbol{v}, \boldsymbol{w}) \in \phi \,\right\}.$$

Similarly, we denote by $\phi^{-1}(S') \subseteq \mathbb{R}^n$ the *preimage* under ϕ of $S' \subseteq \mathbb{R}^m$, that is

$$\phi^{-1}(S') \stackrel{\text{def}}{=} \left\{\, \boldsymbol{v} \in \mathbb{R}^n \;\middle|\; \exists \boldsymbol{w} \in S' \,.\, (\boldsymbol{v}, \boldsymbol{w}) \in \phi \,\right\}.$$

If $n = m$, then the relation ϕ is said to be *space dimension preserving*.

The relation $\phi \subseteq \mathbb{R}^n \times \mathbb{R}^m$ is said to be an *affine relation* if there exists $\ell \in \mathbb{N}$ such that

$$\forall \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{w} \in \mathbb{R}^m : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff \bigwedge_{i=1}^{\ell} \big(\langle \boldsymbol{c}_i, \boldsymbol{w} \rangle \bowtie_i \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle + b_i \big),$$

where $\boldsymbol{a}_i \in \mathbb{R}^n$, $\boldsymbol{c}_i \in \mathbb{R}^m$, $b_i \in \mathbb{R}$ and $\bowtie_i \in \{<, \leq, =, \geq, >\}$, for each $i = 1, \ldots, \ell$.

As a special case, the relation $\phi \subseteq \mathbb{R}^n \times \mathbb{R}^m$ is an *affine function* if and only if there exist a matrix $A \in \mathbb{R}^m \times \mathbb{R}^n$ and a vector $\boldsymbol{b} \in \mathbb{R}^m$ such that,

$$\forall \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{w} \in \mathbb{R}^m : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff \boldsymbol{w} = A\boldsymbol{v} + \boldsymbol{b}.$$

The set $\mathbb{P}_n$ of NNC polyhedra is closed under the application of images and preimages of any space dimension preserving affine relation. The same property holds for the set $\mathbb{CP}_n$ of closed polyhedra, provided the affine relation makes no use of the strict relation symbols $<$ and $>$. Images and preimages of affine relations can be used to model several kinds of transition relations, including deterministic assignments of affine expressions, (affinely constrained) nondeterministic assignments and affine conditional guards.

A space dimension preserving relation $\phi \subseteq \mathbb{R}^n \times \mathbb{R}^n$ can be specified by means of a shorthand notation:

- the vector $\boldsymbol{x} = (x_0, \ldots, x_{n-1})^{\mathrm{T}}$ of *unprimed* variables is used to represent the space dimensions of the domain of ϕ;
- the vector $\boldsymbol{x}' = (x'_0, \ldots, x'_{n-1})^{\mathrm{T}}$ of *primed* variables is used to represent the space dimensions of the range of ϕ;

- any primed variable that "does not occur" in the shorthand specification is meant to be *unaffected* by the relation; namely, for each index $i \in \{0, \ldots, n-1\}$, if in the syntactic specification of the relation the primed variable x'_i only occurs (if ever) with coefficient 0, then it is assumed that the specification also contains the constraint $x'_i = x_i$.

As an example, assuming $\phi \subseteq \mathbb{R}^3 \times \mathbb{R}^3$, the notation $x'_0 - x'_2 \geq 2x_0 - x_1$, where the primed variable x'_1 does not occur, is meant to specify the affine relation defined by

$$\forall \boldsymbol{v} \in \mathbb{R}^3, \boldsymbol{w} \in \mathbb{R}^3 : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff (w_0 - w_2 \geq 2v_0 - v_1) \wedge (w_1 = v_1).$$

The same relation is specified by $x'_0 + 0 \cdot x'_1 - x'_2 \geq 2x_0 - x_1$, since x'_1 occurs with coefficient 0.

The library allows for the computation of images and preimages of polyhedra under restricted subclasses of space dimension preserving affine relations, as described in the following.

1.6.10 Single-Update Affine Functions.

Given a primed variable x'_k and an unprimed affine expression $\langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$, the *affine function* $\phi = \big(x'_k = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b\big) : \mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$\forall \boldsymbol{v} \in \mathbb{R}^n : \phi(\boldsymbol{v}) = A\boldsymbol{v} + \boldsymbol{b},$$

where

$$A = \begin{pmatrix} 1 & & 0 & 0 & \cdots & \cdots & 0 \\ & \ddots & & \vdots & & & \vdots \\ 0 & & 1 & 0 & \cdots & \cdots & 0 \\ a_0 & \cdots & a_{k-1} & a_k & a_{k+1} & \cdots & a_{n-1} \\ 0 & \cdots & \cdots & 0 & 1 & & 0 \\ \vdots & & & \vdots & & \ddots & \\ 0 & \cdots & \cdots & 0 & 0 & & 1 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ b \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and the a_i (resp., b) occur in the $(k+1)$st row in A (resp., position in $\boldsymbol{b}$). Thus function ϕ maps any vector $(v_0, \ldots, v_{n-1})^{\mathrm{T}}$ to

$$\Big(v_0, \ldots, \big(\textstyle\sum_{i=0}^{n-1} a_i v_i + b\big), \ldots, v_{n-1}\Big)^{\mathrm{T}}.$$

The *affine image* operator computes the affine image of a polyhedron $\mathcal{P}$ under $x'_k = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$. For instance, suppose the polyhedron $\mathcal{P}$ to be transformed is the square in $\mathbb{R}^2$ generated by the set of points $\big\{(0,0)^{\mathrm{T}}, (0,3)^{\mathrm{T}}, (3,0)^{\mathrm{T}}, (3,3)^{\mathrm{T}}\big\}$. Then, if the primed variable is x_0 and the affine expression is $x_0 + 2x_1 + 4$ (so that $k = 0$, $a_0 = 1, a_1 = 2, b = 4$), the affine image operator will translate $\mathcal{P}$ to the parallelogram $\mathcal{P}_1$ generated by the set of points $\big\{(4,0)^{\mathrm{T}}, (10,3)^{\mathrm{T}}, (7,0)^{\mathrm{T}}, (13,3)^{\mathrm{T}}\big\}$ with height equal to the side of the square and oblique sides parallel to the line $x_0 - 2x_1$. If the primed variable is as before (i.e., $k = 0$) but the affine expression is x_1 (so that $a_0 = 0, a_1 = 1, b = 0$), then the resulting polyhedron $\mathcal{P}_2$ is the positive diagonal of the square.

The *affine preimage* operator computes the affine preimage of a polyhedron $\mathcal{P}$ under $x'_k = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$. For instance, suppose now that we apply the affine preimage operator as given in the first example using primed variable x_0 and affine expression $x_0 + 2x_1 + 4$ to the parallelogram $\mathcal{P}_1$; then we get the original square $\mathcal{P}$ back. If, on the other hand, we apply the affine preimage operator as given in the second example using primed variable x_0 and affine expression x_1 to $\mathcal{P}_2$, then the resulting polyhedron is the stripe obtained by adding the line $(1,0)^{\mathrm{T}}$ to polyhedron $\mathcal{P}_2$.

Observe that provided the coefficient a_k of the considered variable in the affine expression is non-zero, the affine function is invertible.

1.6.11 Single-Update Bounded Affine Relations.

Given a primed variable x'_k and two unprimed affine expressions $\mathrm{lb} = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$ and $\mathrm{ub} = \langle \boldsymbol{c}, \boldsymbol{x} \rangle + d$, the *bounded affine relation* $\phi = (\mathrm{lb} \leq x'_k \leq \mathrm{ub})$ is defined as

$$\forall \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{w} \in \mathbb{R}^n : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff \big(\langle \boldsymbol{a}, \boldsymbol{v} \rangle + b \leq w_k \leq \langle \boldsymbol{c}, \boldsymbol{v} \rangle + d\big) \wedge \Big(\bigwedge_{0 \leq i < n, i \neq k} w_i = v_i \Big).$$

1.6.12 Generalized Affine Relations.

Similarly, the *generalized affine relation* $\phi = (\mathrm{lhs}' \bowtie \mathrm{rhs})$, where $\mathrm{lhs} = \langle \boldsymbol{c}, \boldsymbol{x} \rangle + d$ and $\mathrm{rhs} = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$ are affine expressions and $\bowtie \in \{<, \leq, =, \geq, >\}$ is a relation symbol, is defined as

$$\forall \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{w} \in \mathbb{R}^n : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff \big(\langle \boldsymbol{c}, \boldsymbol{w} \rangle + d \bowtie \langle \boldsymbol{a}, \boldsymbol{v} \rangle + b\big) \wedge \Big(\bigwedge_{0 \leq i < n, c_i = 0} w_i = v_i \Big).$$

When $\mathrm{lhs} = x_k$ and $\bowtie \in \{=\}$, then the above affine relation becomes equivalent to the single-update affine function $x'_k = \mathrm{rhs}$ (hence the name given to this operator). It is worth stressing that the notation is not symmetric, because the variables occurring in expression lhs are interpreted as primed variables, whereas those occurring in rhs are unprimed; for instance, the transfer relations $\mathrm{lhs}' \leq \mathrm{rhs}$ and $\mathrm{rhs}' \geq \mathrm{lhs}$ are not equivalent in general.

1.6.13 Cylindrification Operator

The operator `unconstrain` computes the *cylindrification* [HMT71] of a polyhedron with respect to one of its variables. Formally, the cylindrification $\mathcal{Q} \in \mathbb{P}_n$ of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ with respect to variable index $i \in \{0, \ldots, n-1\}$ is defined as follows:

$$\mathcal{Q} = \big\{ \boldsymbol{w} \in \mathbb{R}^n \;\big|\; \exists \boldsymbol{v} \in \mathcal{P} \,.\, \forall j \in \{0, \ldots, n-1\} : j \neq i \implies w_j = v_j \big\}.$$

Cylindrification is an idempotent operation; in particular, note that the computed result has the same space dimension of the original polyhedron. A variant of the operator above allows for the cylindrification of a polyhedron with respect to a finite set of variables.

1.6.14 Time-Elapse Operator

The *time-elapse* operator has been defined in [HPR97]. Actually, the time-elapse operator provided by the library is a slight generalization of that one, since it also works on NNC polyhedra. For any two NNC polyhedra $\mathcal{P}, \mathcal{Q} \in \mathbb{P}_n$, the time-elapse between $\mathcal{P}$ and $\mathcal{Q}$, denoted $\mathcal{P} \nearrow \mathcal{Q}$, is the smallest NNC polyhedron containing the set

$$\big\{ \boldsymbol{p} + \lambda \boldsymbol{q} \in \mathbb{R}^n \;\big|\; \boldsymbol{p} \in \mathcal{P}, \boldsymbol{q} \in \mathcal{Q}, \lambda \in \mathbb{R}_+ \big\}.$$

Note that, if $\mathcal{P}, \mathcal{Q} \in \mathbb{CP}_n$ are closed polyhedra, the above set is also a closed polyhedron. In contrast, when $\mathcal{Q}$ is not topologically closed, the above set might not be an NNC polyhedron.

1.6.15 Meet-Preserving Enlargement and Simplification

Let $\mathcal{P}, \mathcal{Q}, \mathcal{R} \in \mathbb{P}_n$ be NNC polyhedra. Then:

- $\mathcal{R}$ is *meet-preserving* with respect to $\mathcal{P}$ using context $\mathcal{Q}$ if $\mathcal{R} \cap \mathcal{Q} = \mathcal{P} \cap \mathcal{Q}$;

- $\mathcal{R}$ is an *enlargement* of $\mathcal{P}$ if $\mathcal{R} \supseteq \mathcal{P}$.
- $\mathcal{R}$ is a *simplification* with respect to $\mathcal{P}$ if $r \leq p$, where r and p are the cardinalities of minimized constraint representations for $\mathcal{R}$ and $\mathcal{P}$, respectively.

Notice that an enlargement need not be a simplification, and vice versa; moreover, the identity function is (trivially) a meet-preserving enlargement and simplification.

The library provides a binary operator (`simplify_using_context`) for the domain of NNC polyhedra that returns a polyhedron which is a meet-preserving enlargement simplification of its first argument using the second argument as context.

The concept of meet-preserving enlargement and simplification also applies to the other basic domains (boxes, grids, BD and octagonal shapes). See below for a definition of the concept of meet-preserving simplification for powerset domains.

1.6.16 Relation-With Operators

The library provides operators for checking the relation holding between an NNC polyhedron and either a constraint or a generator.

Suppose $\mathcal{P}$ is an NNC polyhedron and $\mathcal{C}$ an arbitrary constraint system representing $\mathcal{P}$. Suppose also that $c = \big(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b\big)$ is a constraint with $\bowtie \in \{=, \geq, >\}$ and $\mathcal{Q}$ the set of points that satisfy c. The possible relations between $\mathcal{P}$ and c are as follows.

- $\mathcal{P}$ *is disjoint* from c if $\mathcal{P} \cap \mathcal{Q} = \varnothing$; that is, adding c to $\mathcal{C}$ gives us the empty polyhedron.
- $\mathcal{P}$ *strictly intersects* c if $\mathcal{P} \cap \mathcal{Q} \neq \varnothing$ and $\mathcal{P} \cap \mathcal{Q} \subset \mathcal{P}$; that is, adding c to $\mathcal{C}$ gives us a non-empty polyhedron strictly smaller than $\mathcal{P}$.
- $\mathcal{P}$ *is included* in c if $\mathcal{P} \subseteq \mathcal{Q}$; that is, adding c to $\mathcal{C}$ leaves $\mathcal{P}$ unchanged.
- $\mathcal{P}$ *saturates* c if $\mathcal{P} \subseteq \mathcal{H}$, where $\mathcal{H}$ is the hyperplane induced by constraint c, i.e., the set of points satisfying the equality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$; that is, adding the constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ to $\mathcal{C}$ leaves $\mathcal{P}$ unchanged.

The polyhedron $\mathcal{P}$ *subsumes* the generator g if adding g to any generator system representing $\mathcal{P}$ does not change $\mathcal{P}$.

1.6.17 Widening Operators

The library provides two widening operators for the domain of polyhedra. The first one, that we call *H79-widening*, mainly follows the specification provided in the PhD thesis of N. Halbwachs [Hal79], also described in [HPR97]. Note that in the computation of the H79-widening $\mathcal{P} \nabla \mathcal{Q}$ of two polyhedra $\mathcal{P}, \mathcal{Q} \in \mathbb{CP}_n$ it is required as a precondition that $\mathcal{P} \subseteq \mathcal{Q}$ (the same assumption was implicitly present in the cited papers).

The second widening operator, that we call *BHRZ03-widening*, is an instance of the specification provided in [BHRZ03a]. This operator also requires as a precondition that $\mathcal{P} \subseteq \mathcal{Q}$ and it is guaranteed to provide a result which is at least as precise as the H79-widening.

Both widening operators can be applied to NNC polyhedra. The user is warned that, in such a case, the results may not closely match the geometric intuition which is at the base of the specification of the two widenings. The reason is that, in the current implementation, the widenings are not directly applied to the NNC polyhedra, but rather to their internal representations. Implementation work is in progress and future versions of the library may provide an even better integration of the two widenings with the domain of NNC polyhedra.

Note

As is the case for the other operators on polyhedra, the implementation overwrites one of the two polyhedra arguments with the result of the widening application. To avoid trivial misunderstandings, it is worth stressing that if polyhedra $\mathcal{P}$ and $\mathcal{Q}$ (where $\mathcal{P} \subseteq \mathcal{Q}$) are identified by program variables `p` and `q`, respectively, then the call `q.H79_widening_assign(p)` will assign the polyhedron $\mathcal{P} \nabla \mathcal{Q}$ to variable `q`. Namely, it is the bigger polyhedron $\mathcal{Q}$ which is overwritten by the result of the widening. The smaller polyhedron is not modified, so as to lead to an easier coding of the usual convergence test ($\mathcal{P} \supseteq \mathcal{P} \nabla \mathcal{Q}$ can be coded as `p.contains(q)`). Note that, in the above context, a call such as `p.H79_widening_assign(q)` is likely to result in undefined behavior, since the precondition $\mathcal{Q} \subseteq \mathcal{P}$ will be missed (unless it happens that $\mathcal{P} = \mathcal{Q}$). The same observation holds for all flavors of widenings and extrapolation operators that are implemented in the library and for all the language interfaces.

1.6.18 Widening with Tokens

When approximating a fixpoint computation using widening operators, a common tactic to improve the precision of the final result is to delay the application of widening operators. The usual approach is to fix a parameter k and only apply widenings starting from the k-th iteration.

The library also supports an improved widening delay strategy, that we call *widening with tokens* [BHRZ03a]. A token is a sort of wild card allowing for the replacement of the widening application by the exact upper bound computation: the token is used (and thus consumed) only when the widening would have resulted in an actual precision loss (as opposed to the *potential* precision loss of the classical delay strategy). Thus, all widening operators can be supplied with an optional argument, recording the number of available tokens, which is decremented when tokens are used. The approximated fixpoint computation will start with a fixed number k of tokens, which will be used if and when needed. When there are no tokens left, the widening is always applied.

1.6.19 Extrapolation Operators

Besides the two widening operators, the library also implements several *extrapolation* operators, which differ from widenings in that their use along an upper iteration sequence does not ensure convergence in a finite number of steps.

In particular, for each of the two widenings there is a corresponding *limited* extrapolation operator, which can be used to implement the *widening "up to"* technique as described in [HPR97]. Each limited extrapolation operator takes a constraint system as an additional parameter and uses it to improve the approximation yielded by the corresponding widening operator. Note that a convergence guarantee can only be obtained by suitably restricting the set of constraints that can occur in this additional parameter. For instance, in [HPR97] this set is fixed once and for all before starting the computation of the upward iteration sequence.

The *bounded* extrapolation operators further enhance each one of the limited extrapolation operators described above by intersecting the result of the limited extrapolation operation with the box obtained as a result of applying the CC76-widening to the smallest boxes enclosing the two argument polyhedra.

1.7 Intervals and Boxes

The PPL provides support for computations on non-relational domains, called boxes, and also the interval domains used for their representation.

An *interval* in $\mathbb{R}$ is a pair of *bounds*, called *lower* and *upper*. Each bound can be either (1) *closed and bounded*, (2) *open and bounded*, or (3) *open and unbounded*. If the bound is *bounded*, then it has a value in $\mathbb{R}$. For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, and for each relation symbol $\bowtie \in \{=, \geq, >\}$, the

constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ is said to be a *interval constraint* if there exist an index $i \in \{0, \ldots, n-1\}$ such that, for all $k \in \{0, \ldots, i-1, i+1, \ldots, n-1\}$, $a_k = 0$. Thus each interval constraint that is not a tautology or inconsistent has the form $x = r$, $x \leq r$, $x \geq r$, $x < r$ or $x > r$, with $r \in \mathbb{R}$.

Letting $\mathcal{B}$ be a sequence of n intervals and $\boldsymbol{e}_i = (0, \ldots, 1, \ldots, 0)^{\mathrm{T}}$ be the vector in $\mathbb{R}^n$ with 1 in the i'th position and zeroes in every other position; if the lower bound of the i'th interval in $\mathcal{B}$ is bounded, the corresponding interval constraint is defined as $\langle \boldsymbol{e}_i, \boldsymbol{x} \rangle \bowtie b$, where b is the value of the bound and $\bowtie$ is $\geq$ if it is a closed bound and $>$ if it is an open bound. Similarly, if the upper bound of the i'th interval in $\mathcal{B}$ is bounded, the corresponding interval constraint is defined as $\langle \boldsymbol{e}_i, \boldsymbol{x} \rangle \bowtie b$, where b is the value of the bound and $\bowtie$ is $\leq$ if it is a closed bound and $<$ if it is an open bound.

A convex polyhedron $\mathcal{P} \in \mathbb{CP}_n$ is said to be a *box* if and only if either $\mathcal{P}$ is the set of solutions to a finite set of interval constraints or $n = 0$ and $\mathcal{P} = \varnothing$. Therefore any n-dimensional *box* $\mathcal{P}$ in $\mathbb{R}^n$ where $n > 0$ can be represented by a sequence of n intervals $\mathcal{B}$ in $\mathbb{R}$ and $\mathcal{P}$ is a closed polyhedron if every bound in the intervals in $\mathcal{B}$ is either closed and bounded or open and unbounded.

1.7.1 Widening and Extrapolation Operators on Boxes

The library provides a widening operator for boxes. Given two sequences of intervals defining two n-dimensional boxes, the *CC76-widening* applies, for each corresponding interval and bound, the interval constraint widening defined in [CC76]. For extra precision, this incorporates the widening with thresholds as defined in [BCCetal02] with $\{-2, -1, 0, 1, 2\}$ as the set of default threshold values.

1.8 Weakly-Relational Shapes

The PPL provides support for computations on numerical domains that, in selected contexts, can achieve a better precision/efficiency ratio with respect to the corresponding computations on a "fully relational" domain of convex polyhedra. This is achieved by restricting the syntactic form of the constraints that can be used to describe the domain elements.

1.8.1 Bounded Difference Shapes

For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, and for each relation symbol $\bowtie \in \{=, \geq\}$, the linear constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ is said to be a *bounded difference* if there exist two indices $i, j \in \{0, \ldots, n-1\}$ such that:

- $a_i, a_j \in \{-1, 0, 1\}$ and $a_i \neq a_j$;
- $a_k = 0$, for all $k \notin \{i, j\}$.

A convex polyhedron $\mathcal{P} \in \mathbb{CP}_n$ is said to be a *bounded difference shape* (BDS, for short) if and only if either $\mathcal{P}$ can be expressed as the intersection of a finite number of bounded difference constraints or $n = 0$ and $\mathcal{P} = \varnothing$.

1.8.2 Octagonal Shapes

For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, and for each relation symbol $\bowtie \in \{=, \geq\}$, the linear constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ is said to be an *octagonal* if there exist two indices $i, j \in \{0, \ldots, n-1\}$ such that:

- $a_i, a_j \in \{-1, 0, 1\}$;
- $a_k = 0$, for all $k \notin \{i, j\}$.

A convex polyhedron $\mathcal{P} \in \mathbb{CP}_n$ is said to be an *octagonal shape* (OS, for short) if and only if either $\mathcal{P}$ can be expressed as the intersection of a finite number of octagonal constraints or $n = 0$ and $\mathcal{P} = \varnothing$.

Note that, since any bounded difference is also an octagonal constraint, any BDS is also an OS. The name "octagonal" comes from the fact that, in a vector space of dimension 2, a bounded OS can have eight sides at most.

1.8.3 Weakly-Relational Shapes Interface

By construction, any BDS or OS is always topologically closed. Under the usual set inclusion ordering, the set of all BDSs (resp., OSs) on the vector space $\mathbb{R}^n$ is a lattice having the empty set $\varnothing$ and the universe $\mathbb{R}^n$ as the smallest and the biggest elements, respectively. In theoretical terms, it is a meet sub-lattice of $\mathbb{CP}_n$; moreover, the lattice of BDSs is a meet sublattice of the lattice of OSs. The least upper bound of a finite set of BDSs (resp., OSs) is said to be their *bds-hull* (resp., *oct-hull*).

As far as the representation of the rational inhomogeneous term of each bounded difference or octagonal constraint is concerned, several *rounding-aware* implementation choices are available, including:

- bounded precision integer types;
- bounded precision floating point types;
- unbounded precision integer and rational types, as provided by GMP.

The user interface for BDSs and OSs is meant to be as similar as possible to the one developed for the domain of closed polyhedra: in particular, all operators on polyhedra are also available for the domains of BDSs and OSs, even though they are typically characterized by a lower degree of precision. For instance, the *bds-difference* and *oct-difference* operators return (the smallest) over-approximations of the set-theoretical difference operator on the corresponding domains. In the case of (generalized) images and preimages of affine relations, suitable (possibly not-optimal) over-approximations are computed when the considered relations cannot be precisely modeled by only using bounded differences or octagonal constraints.

1.8.4 Widening and Extrapolation Operators on Weakly-Relational Shapes

For the domains of BDSs and OSs, the library provides a variant of the widening operator for convex polyhedra defined in [CH78]. The implementation follows the specification in [BHMZ05a,BHMZ05b], resulting in an operator which is well-defined on the corresponding domain (i.e., it does not depend on the internal representation of BDSs or OSs), while still ensuring convergence in a finite number of steps.

The library also implements an extension of the widening operator for intervals as defined in [CC76]. The reader is warned that such an extension, even though being well-defined on the domain of BDSs and OSs, is not provided with a convergence guarantee and is therefore an extrapolation operator.

1.9 Rational Grids

In this section we introduce rational grids as provided by the library. See also [BDHetal05] for a detailed description of this domain.

The library supports two representations for the grids domain; *congruence systems* and *grid generator systems*. We first describe *linear congruence relations* which form the elements of a congruence system.

1.9.1 Congruences and Congruence Relations

For any $a, b, f \in \mathbb{R}$, $a \equiv_f b$ denotes the *congruence* $\exists \mu \in \mathbb{Z} \,.\, a - b = \mu f$.

Let $\mathbb{S} \in \{\mathbb{Q}, \mathbb{R}\}$. For each vector $\boldsymbol{a} \in \mathbb{S}^n \setminus \{\boldsymbol{0}\}$ and scalars $b, f \in \mathbb{S}$, the notation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$ stands for the *linear congruence relation in* $\mathbb{S}^n$ defined by the set of vectors

$$\left\{ \boldsymbol{v} \in \mathbb{R}^n \;\middle|\; \exists \mu \in \mathbb{Z} \,.\, \langle \boldsymbol{a}, \boldsymbol{v} \rangle = b + \mu f \right\};$$

when $f \neq 0$, the relation is said to be *proper*; $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_0 b$ (i.e., when $f = 0$) denotes the equality $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$. f is called the *frequency* or *modulus* and b the *base value* of the relation. Thus, provided $\boldsymbol{a} \neq \boldsymbol{0}$, the relation $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b$ defines the set of affine hyperplanes

$$\left\{ \left(\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b + \mu f \right) \;\middle|\; \mu \in \mathbb{Z} \right\};$$

if $b \equiv_f 0$, $\langle \boldsymbol{0}, \boldsymbol{x} \rangle \equiv_f b$ defines the universe $\mathbb{R}^n$ and the empty set, otherwise.

1.9.2 Rational Grids

The set $\mathcal{L} \subseteq \mathbb{R}^n$ is a *rational grid* if and only if either $\mathcal{L}$ is the set of vectors in $\mathbb{R}^n$ that satisfy a finite system $\mathcal{C}$ of congruence relations in $\mathbb{Q}^n$ or $n = 0$ and $\mathcal{L} = \varnothing$.

We also say that $\mathcal{L}$ *is described by* $\mathcal{C}$ and that $\mathcal{C}$ *is a congruence system for* $\mathcal{L}$.

The *grid domain* $\mathbb{G}_n$ is the set of all rational grids described by finite sets of congruence relations in $\mathbb{Q}^n$.

If the congruence system $\mathcal{C}$ describes the $\varnothing$, the *empty* grid, then we say that $\mathcal{C}$ is *inconsistent*. For example, the congruence systems $\left\{ \langle \boldsymbol{0}, \boldsymbol{x} \rangle \equiv_0 1 \right\}$ meaning that $0 = 1$ and $\left\{ \langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_2 0, \langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_2 1 \right\}$, for any $\boldsymbol{a} \in \mathbb{R}^n$, meaning that the value of an expression must be both even and odd are both inconsistent since both describe the empty grid.

When ordering grids by the set inclusion relation, the empty set $\varnothing$ and the vector space $\mathbb{R}^n$ (which is described by the empty set of congruence relations) are, respectively, the smallest and the biggest elements of $\mathbb{G}_n$. The vector space $\mathbb{R}^n$ is also called the *universe* grid.

In set theoretical terms, $\mathbb{G}_n$ is a *lattice* under set inclusion.

1.9.3 Integer Combinations

Let $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} \subseteq \mathbb{R}^n$ be a finite set of vectors. For all scalars $\mu_1, \ldots, \mu_k \in \mathbb{Z}$, the vector $\boldsymbol{v} = \sum_{j=1}^{k} \mu_j \boldsymbol{x}_j$ is said to be a *integer* combination of the vectors in S.

We denote by $\mathrm{int.hull}(S)$ (resp., $\mathrm{int.affine.hull}(S)$) the set of all the integer (resp., integer and affine) combinations of the vectors in S.

1.9.4 Points, Parameters and Lines

Let $\mathcal{L}$ be a grid. Then

- a vector $\boldsymbol{p} \in \mathcal{L}$ is called a *grid point* of $\mathcal{L}$;
- a vector $\boldsymbol{q} \in \mathbb{R}^n$, where $\boldsymbol{q} \neq \boldsymbol{0}$, is called a *parameter* of $\mathcal{L}$ if $\mathcal{L} \neq \varnothing$ and $\boldsymbol{p} + \mu \boldsymbol{q} \in \mathcal{L}$, for all points $\boldsymbol{p} \in \mathcal{L}$ and all $\mu \in \mathbb{Z}$;
- a vector $\boldsymbol{l} \in \mathbb{R}^n$ is called a *grid line* of $\mathcal{L}$ if $\mathcal{L} \neq \varnothing$ and $\boldsymbol{p} + \lambda \boldsymbol{l} \in \mathcal{L}$, for all points $\boldsymbol{p} \in \mathcal{L}$ and all $\lambda \in \mathbb{R}$.

1.9.5 The Grid Generator Representation

We can generate any rational grid in $\mathbb{G}_n$ from a finite subset of its points, parameters and lines; each point in a grid is obtained by adding a linear combination of its generating lines to an integral combination of its parameters and an integral affine combination of its generating points.

If L, Q, P are each finite subsets of $\mathbb{Q}^n$ and

$$\mathcal{L} = \text{linear.hull}(L) + \text{int.hull}(Q) + \text{int.affine.hull}(P)$$

where the symbol '$+$' denotes the Minkowski's sum, then $\mathcal{L} \in \mathbb{G}_n$ is a rational grid (see Section 4.4 in [Sch99] and also Proposition 8 in [BDHetal05]). The 3-tuple (L, Q, P) is said to be a *grid generator system* for $\mathcal{L}$ and we write $\mathcal{L} = \text{ggen}(L, Q, P)$.

Note that the grid $\mathcal{L} = \text{ggen}(L, Q, P) = \varnothing$ if and only if the set of grid points $P = \varnothing$. If $P \neq \varnothing$, then $\mathcal{L} = \text{ggen}(L, \varnothing, Q_{\boldsymbol{p}} \cup P)$ where, for some $\boldsymbol{p} \in P$, $Q_{\boldsymbol{p}} = \{\, \boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{q} \in Q \,\}$.

1.9.6 Minimized Grid Representations

A *minimized* congruence system $\mathcal{C}$ for $\mathcal{L}$ is such that, if $\mathcal{C}'$ is another congruence system for $\mathcal{L}$, then $\#\mathcal{C} \leq \#\mathcal{C}'$. Note that a minimized congruence system for a non-empty grid has at most n congruence relations.

Similarly, a *minimized* grid generator system $\mathcal{G} = (L, Q, P)$ for $\mathcal{L}$ is such that, if $\mathcal{G}' = (L', Q', P')$ is another grid generator system for $\mathcal{L}$, then $\#L \leq \#L'$ and $\#Q + \#P \leq \#Q' + \#P'$. Note that a minimized grid generator system for a grid has no more than a total of $n + 1$ grid lines, parameters and points.

1.9.7 Double Description for Grids

As for convex polyhedra, any grid $\mathcal{L}$ can be described by using a congruence system $\mathcal{C}$ for $\mathcal{L}$, a grid generator system $\mathcal{G}$ for $\mathcal{L}$, or both by means of the *double description pair (DD pair)* $(\mathcal{C}, \mathcal{G})$. The *double description method* for grids is a collection of theoretical results very similar to those for convex polyhedra showing that, given one kind of representation, there are algorithms for computing a representation of the other kind and for minimizing both representations.

As for convex polyhedra, such changes of representation form a key step in the implementation of many operators on grids such as, for example, intersection and grid join.

1.9.8 Space Dimensions and Dimension-compatibility for Grids

The *space dimension* of a grid $\mathcal{L} \in \mathbb{G}_n$ is the dimension $n \in \mathbb{N}$ of the corresponding vector space $\mathbb{R}^n$. The space dimension of congruence relations, grid generators and other objects of the library is defined similarly.

1.9.9 Affine Independence and Affine Dimension for Grids

A *non-empty* grid $\mathcal{L} \in \mathbb{G}_n$ has *affine dimension* $k \in \mathbb{N}$, denoted by $\dim(\mathcal{G}) = k$, if the maximum number of affinely independent points in $\mathcal{G}$ is $k + 1$. The affine dimension of an empty grid is defined to be 0. Thus we have $0 \leq \dim(\mathcal{G}) \leq n$.

1.10 Operations on Rational Grids

In general, the operations on rational grids are the same as those for the other PPL domains and the definitions of these can be found in Section Operations on Convex Polyhedra. Below we just describe those operations that have features or behavior that is in some way special to the grid domain.

1.10.1 Affine Images and Preimages

As for convex polyhedra (see Single-Update Affine Functions), the library provides affine image and preimage operators for grids: given a variable x_k and linear expression $\mathrm{expr} = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$, these determine the affine transformation $\phi = \big(x'_k = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b\big) \colon \mathbb{R}^n \to \mathbb{R}^n$ that transforms any point $(v_0, \ldots, v_{n-1})^{\mathrm{T}}$ in a grid $\mathcal{L}$ to

$$\Big(v_0, \ldots, \big(\textstyle\sum_{i=0}^{n-1} a_i v_i + b\big), \ldots, v_{n-1}\Big)^{\mathrm{T}}.$$

The *affine image* operator computes the affine image of a grid $\mathcal{L}$ under $x'_k = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$. For instance, suppose the grid $\mathcal{L}$ to be transformed is the non-relational grid in $\mathbb{R}^2$ generated by the set of grid points $\big\{(0,0)^{\mathrm{T}}, (0,3)^{\mathrm{T}}, (3,0)^{\mathrm{T}}\big\}$. Then, if the considered variable is x_0 and the linear expression is $3x_0 + 2x_1 + 1$ (so that $k = 0$, $a_0 = 3, a_1 = 2, b = 1$), the affine image operator will translate $\mathcal{L}$ to the grid $\mathcal{L}_1$ generated by the set of grid points $\big\{(1,0)^{\mathrm{T}}, (7,3)^{\mathrm{T}}, (10,0)^{\mathrm{T}}\big\}$ which is the grid generated by the grid point $(1,0)$ and parameters $(3,-3), (0,9)$; or, alternatively defined by the congruence system $\{x \equiv_3 1, x + y \equiv_9 1\}$. If the considered variable is as before (i.e., $k = 0$) but the linear expression is x_1 (so that $a_0 = 0, a_1 = 1, b = 0$), then the resulting grid $\mathcal{L}_2$ is the grid containing all the points whose coordinates are integral multiples of 3 and lie on line $x = y$.

The affine preimage operator computes the affine preimage of a grid $\mathcal{L}$ under ϕ. For instance, suppose now that we apply the affine preimage operator as given in the first example using variable x_0 and linear expression $3x_0 + 2x_1 + 1$ to the grid $\mathcal{L}_1$; then we get the original grid $\mathcal{L}$ back. If, on the other hand, we apply the affine preimage operator as given in the second example using variable x_0 and linear expression x_1 to $\mathcal{L}_2$, then the resulting grid will consist of all the points in $\mathbb{R}^2$ where the y coordinate is an integral multiple of 3.

Observe that provided the coefficient a_k of the considered variable in the linear expression is non-zero, the affine transformation is invertible.

1.10.2 Generalized Affine Images

Similarly to convex polyhedra (see Generalized Affine Relations), the library provides two other grid operators that are generalizations of the single update affine image and preimage operators for grids. The *generalized affine image* operator $\phi = (\mathrm{lhs}', \mathrm{rhs}, f) \colon \mathbb{R}^n \to \mathbb{R}^n$, where $\mathrm{lhs} = \langle \boldsymbol{c}, \boldsymbol{x} \rangle + d$ and $\mathrm{rhs} = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$ are affine expressions and $f \in \mathbb{Q}$, is defined as

$$\forall \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{w} \in \mathbb{R}^n : (\boldsymbol{v}, \boldsymbol{w}) \in \phi \iff \big(\langle \boldsymbol{c}, \boldsymbol{w} \rangle + d \equiv_f \langle \boldsymbol{a}, \boldsymbol{v} \rangle + b\big) \wedge \Big(\bigwedge_{0 \le i < n, c_i = 0} w_i = v_i \Big).$$

Note that, when $\mathrm{lhs} = x_k$ and $f = 0$, so that the transfer function is an equality, then the above operator is equivalent to the application of the standard affine image of $\mathcal{L}$ with respect to the variable x_k and the affine expression rhs.

1.10.3 Frequency Operator

Let $\mathcal{L} \in \mathbb{G}_n$ be any non-empty grid and $\mathrm{expr} = \big(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + b\big)$ be a linear expression. Then if, for some $c, f \in \mathbb{R}$, all the points in $\mathcal{L}$ satisfy the congruence $\mathrm{cg} = (\mathrm{expr} \equiv_f c)$, then the maximum f such that this holds is called the *frequency* of $\mathcal{L}$ with respect to expr.

The frequency operator provided by the library returns both the frequency f and a value $\mathrm{val} = \langle \boldsymbol{a}, \boldsymbol{w} \rangle + b$ where $\boldsymbol{w} \in \mathcal{L}$ and

$$|\mathrm{val}| = \min\Big\{ \big|\langle \boldsymbol{a}, \boldsymbol{v} \rangle + b\big| \;\Big|\; \boldsymbol{v} \in \mathcal{L} \Big\}.$$

Observe that the above definition is also applied to other simple objects in the library like polyhedra, octagonal shapes, bd-shapes and boxes and in such cases the definition of frequency can be simplified. For instance, the frequency for an object $\mathcal{P} \in \mathbb{P}_n$ is defined if and only if there is a unique value c such that $\mathcal{P}$ saturates the equality $(\mathrm{expr} = c)$; in this case the frequency is 0 and the value returned is c.

1.10.4 Time-Elapse Operator

For any two grids $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{G}_n$, the *time-elapse* between $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_1 \nearrow \mathcal{L}_2$, is the grid

$$\big\{ \boldsymbol{p} + \mu \boldsymbol{q} \in \mathbb{R}^n \;\big|\; \boldsymbol{p} \in \mathcal{L}_1, \boldsymbol{q} \in \mathcal{L}_2, \mu \in \mathbb{Z} \big\}.$$

1.10.5 Relation-with Operators

The library provides operators for checking the relation holding between a grid and a congruence, a grid generator, a constraint or a (polyhedron) generator.

Suppose $\mathcal{L}$ is a grid and $\mathcal{C}$ an arbitrary congruence system representing $\mathcal{L}$. Suppose also that $\mathrm{cg} = \big(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \equiv_f b\big)$ is a congruence relation with $\mathcal{L}_{\mathrm{cg}} = \mathrm{gcon}\big(\{\mathrm{cg}\}\big)$. The possible relations between $\mathcal{L}$ and cg are as follows.

- $\mathcal{L}$ *is disjoint* from cg if $\mathcal{L} \cap \mathcal{L}_{\mathrm{cg}} = \varnothing$; that is, adding cg to $\mathcal{C}$ gives us the empty grid.
- $\mathcal{L}$ *strictly intersects* cg if $\mathcal{L} \cap \mathcal{L}_{\mathrm{cg}} \neq \varnothing$ and $\mathcal{L} \cap \mathcal{L}_{\mathrm{cg}} \subset \mathcal{L}$; that is, adding cg to $\mathcal{C}$ gives us a non-empty grid strictly smaller than $\mathcal{L}$.
- $\mathcal{L}$ *is included* in cg if $\mathcal{L} \subseteq \mathcal{L}_{\mathrm{cg}}$; that is, adding cg to $\mathcal{C}$ leaves $\mathcal{L}$ unchanged.
- $\mathcal{L}$ *saturates* cg if $\mathcal{L}$ *is included* in cg and $f = 0$, i.e., cg is an equality congruence.

For the relation between $\mathcal{L}$ and a constraint, suppose that $c = \big(\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b\big)$ is a constraint with $\bowtie \in \{=, \geq, >\}$ and $\mathcal{Q}$ the set of points that satisfy c. The possible relations between $\mathcal{L}$ and c are as follows.

- $\mathcal{L}$ *is disjoint* from c if $\mathcal{L} \cap \mathcal{Q} = \varnothing$.
- $\mathcal{L}$ *strictly intersects* c if $\mathcal{L} \cap \mathcal{Q} \neq \varnothing$ and $\mathcal{L} \cap \mathcal{Q} \subset \mathcal{L}$.
- $\mathcal{L}$ *is included* in c if $\mathcal{L} \subseteq \mathcal{Q}$.
- $\mathcal{L}$ *saturates* c if $\mathcal{L}$ *is included* in c and $\bowtie$ is $=$.

A grid $\mathcal{L}$ *subsumes* a grid generator g if adding g to any grid generator system representing $\mathcal{L}$ does not change $\mathcal{L}$.

A grid $\mathcal{L}$ *subsumes* a (polyhedron) point or closure point g if adding the corresponding grid point to any grid generator system representing $\mathcal{L}$ does not change $\mathcal{L}$. A grid $\mathcal{L}$ *subsumes* a (polyhedron) ray or line g if adding the corresponding grid line to any grid generator system representing $\mathcal{L}$ does not change $\mathcal{L}$.

1.10.6 Wrapping Operator

The operator `wrap_assign` provided by the library, allows for the wrapping of a subset of the set of space dimensions so as to fit the given bounded integer type and have the specified overflow behavior. In order to maximize the precision of this operator for grids, the exact behavior differs in some respects from the other simple classes of geometric descriptors.

Suppose $\mathcal{L} \in \mathbb{G}_n$ is a grid and J a subset of the set of space dimensions $\{0, \ldots, n-1\}$. Suppose also that the width of the bounded integer type is w so that the range of values $R = \{r \in \mathbb{R} \mid 0 \leq r < 2^w\}$ if the type is unsigned and $R = \{r \in \mathbb{R} \mid -2^{w-1} \leq r < 2^{w-1}\}$ otherwise. Consider a space dimension $j \in J$ and a variable v_j for dimension j.

If the value in $\mathcal{L}$ for the variable v_j is a constant in the range R, then it is unchanged. Otherwise the result $\mathcal{L}'$ of the operation on $\mathcal{L}$ will depend on the specified overflow behavior.

- Overflow impossible. In this case, it is known that no wrapping can occur. If the grid $\mathcal{L}$ has no value for the variable v_j in the range R, then $\mathcal{L}$ is set empty. If v_j has exactly one value $a \in R$ in $\mathcal{L}$, then v_j is set equal to a. Otherwise, $\mathcal{L}' = \mathcal{L}$.

- Overflow undefined. In this case, for each value a for v_j in the grid $\mathcal{L}$, the wrapped value can be any value $a + z \in R$ where $z \in \mathbb{Z}$. Therefore $\mathcal{L}'$ is obtained by adding the parameter $(0, \ldots, 0, v_j, 0, \ldots, 0)$, where $v_j = 1$, to the generator system for $\mathcal{L}$.

- Overflow wraps. In this case, if $\mathcal{L}$ already satisfies the congruence $v_j = a \mod 2^w$, for some $a \in \mathbb{R}$, then v_j is set equal to a' where $a' = a \mod 2^w$ and $a' \in R$. Otherwise, $\mathcal{L}'$ is obtained by adding the parameter $(0, \ldots, 0, v_j, 0, \ldots, 0)$, where $v_j = 2^w$, to the generator system for $\mathcal{L}$.

1.10.7 Widening Operators

The library provides *grid widening* operators for the domain of grids. The congruence widening and generator widening follow the specifications provided in [BDHetal05]. The third widening uses either the congruence or the generator widening, the exact rule governing this choice at the time of the call is left to the implementation. Note that, as for the widenings provided for convex polyhedra, all the operations provided by the library for computing a widening $\mathcal{L}_1 \nabla \mathcal{L}_2$ of grids $\mathcal{L}_1, \mathcal{L}_2 \in \mathbb{G}_n$ require as a precondition that $\mathcal{L}_1 \subseteq \mathcal{L}_2$.

Note

As is the case for the other operators on grids, the implementation overwrites one of the two grid arguments with the result of the widening application. It is worth stressing that, in any widening operation that computes the widening $\mathcal{L}_1 \nabla \mathcal{L}_2$, the resulting grid will be assigned to overwrite the store containing the bigger grid $\mathcal{L}_2$. The smaller grid $\mathcal{L}_1$ is not modified. The same observation holds for all flavors of widenings and extrapolation operators that are implemented in the library and for all the language interfaces.

1.10.8 Widening with Tokens

This is as for widening with tokens for convex polyhedra.

1.10.9 Extrapolation Operators

Besides the widening operators, the library also implements several *extrapolation* operators, which differ from widenings in that their use along an upper iteration sequence does not ensure convergence in a finite number of steps.

In particular, for each grid widening that is provided, there is a corresponding *limited* extrapolation operator, which can be used to implement the *widening "up to"* technique as described in [HPR97]. Each limited extrapolation operator takes a congruence system as an additional parameter and uses it to improve the approximation yielded by the corresponding widening operator. Note that, as in the case for convex polyhedra, a convergence guarantee can only be obtained by suitably restricting the set of congruence relations that can occur in this additional parameter.

1.11 The Powerset Construction

The PPL provides the finite powerset construction; this takes a pre-existing domain and upgrades it to one that can represent disjunctive information (by using a *finite* number of disjuncts). The construction follows the approach described in [Bag98], also summarized in [BHZ04] where there is an account of generic widenings for the powerset domain (some of which are supported in the pointset powerset domain instantiation of this construction described in Section The Pointset Powerset Domain).

1.11.1 The Powerset Domain

The domain is built from a pre-existing base-level domain D which must include an entailment relation '$\vdash$', meet operation '$\otimes$', a top element '**1**' and bottom element '**0**'.

A set $\mathcal{S} \in \wp(D)$ is called *non-redundant* with respect to '$\vdash$' if and only if $\mathbf{0} \notin \mathcal{S}$ and $\forall d_1, d_2 \in \mathcal{S} : d_1 \vdash d_2 \implies d_1 = d_2$. The set of finite non-redundant subsets of D (with respect to '$\vdash$') is denoted by $\wp^{\vdash}_{\mathrm{fn}}(D)$. The function $\Omega^{\vdash}_{D} \colon \wp_{\mathrm{f}}(D) \to \wp^{\vdash}_{\mathrm{fn}}(D)$, called *Omega-reduction*, maps a finite set into its non-redundant counterpart; it is defined, for each $\mathcal{S} \in \wp_{\mathrm{f}}(D)$, by

$$\Omega^{\vdash}_{D}(\mathcal{S}) \stackrel{\mathrm{def}}{=} \mathcal{S} \setminus \{\, d \in \mathcal{S} \mid d = \mathbf{0} \text{ or } \exists d' \in \mathcal{S} \,.\, d \Vdash d' \,\}.$$

where $d \Vdash d'$ denotes $d \vdash d' \wedge d \neq d'$.

As the intended semantics of a powerset domain element $\mathcal{S} \in \wp_{\mathrm{f}}(D)$ is that of disjunction of the semantics of D, the finite set $\mathcal{S}$ is semantically equivalent to the non-redundant set $\Omega^{\vdash}_{D}(\mathcal{S})$; and elements of $\mathcal{S}$ will be called *disjuncts*. The restriction to the finite subsets reflects the fact that here disjunctions are implemented by explicit collections of disjuncts. As a consequence of this restriction, for any $\mathcal{S} \in \wp_{\mathrm{f}}(D)$ such that $\mathcal{S} \neq \{\mathbf{0}\}$, $\Omega^{\vdash}_{D}(\mathcal{S})$ is the (finite) set of the maximal elements of $\mathcal{S}$.

The *finite powerset domain* over a domain D is the set of all finite non-redundant sets of D and denoted by D_{P}. The domain includes an approximation ordering '$\vdash_{\mathrm{P}}$' defined so that, for any $\mathcal{S}_1$ and $\mathcal{S}_2 \in D_{\mathrm{P}}$, $\mathcal{S}_1 \vdash_{\mathrm{P}} \mathcal{S}_2$ if and only if

$$\forall d_1 \in \mathcal{S}_1 : \exists d_2 \in \mathcal{S}_2 \,.\, d_1 \vdash d_2.$$

Therefore the top element is $\{\mathbf{1}\}$ and the bottom element is the emptyset.

Note

> As far as Omega-reduction is concerned, the library adopts a *lazy* approach: an element of the powerset domain is represented by a potentially redundant sequence of disjuncts. Redundancies can be eliminated by explicitly invoking the operator `omega_reduce()`, e.g., before performing the output of a powerset element. Note that all the documented operators automatically perform Omega-reductions on their arguments, when needed or appropriate.

1.12 Operations on the Powerset Construction

In this section we briefly describe the generic operations on Powerset Domains that are provided by the library for any given base-level domain D.

1.12.1 Meet and Upper Bound

Given the sets $\mathcal{S}_1$ and $\mathcal{S}_2 \in D_{\mathrm{P}}$, the *meet* and *upper bound* operators provided by the library returns the set $\Omega^{\vdash}_{D}\big(\{\, d_1 \otimes d_2 \mid d_1 \in \mathcal{S}_1, d_2 \in \mathcal{S}_2 \,\}\big)$ and Omega-reduced set union $\Omega^{\vdash}_{D}(\mathcal{S}_1 \cup \mathcal{S}_2)$ respectively.

1.12.2 Adding a Disjunct

Given the powerset element $\mathcal{S} \in D_{\mathrm{P}}$ and the base-level element $d \in D$, the *add disjunct* operator provided by the library returns the powerset element $\Omega^{\vdash}_{D}\big(\mathcal{S} \cup \{d\}\big)$.

1.12.3 Collapsing a Powerset Element

If the given powerset element is not empty, then the *collapse* operator returns the singleton powerset consisting of an upper-bound of all the disjuncts.

1.13 The Pointset Powerset Domain

The pointset powerset domain provided by the PPL is the finite powerset domain (defined in Section The Powerset Construction) whose base-level domain D is one of the classes of semantic geometric descriptors listed in Section Semantic Geometric Descriptors.

In addition to the operations described for the generic powerset domain in Section Operations on the Powerset Construction, the PPL provides all the generic operations listed in Generic Operations on Semantic Geometric Descriptors. Here we just describe those operations that are particular to the pointset powerset domain.

1.13.1 Meet-Preserving Simplification

Let $\mathcal{S}_1 = \{d_1, \ldots, d_m\}$, $\mathcal{S}_2 = \{c_1, \ldots, c_n\}$ and $\mathcal{S} = \{s_1, \ldots, s_q\}$ be Omega-reduced elements of a pointset powerset domain over the same base-level domain. Then:

- $\mathcal{S}$ is *powerset meet-preserving* with respect to $\mathcal{S}_1$ using context $\mathcal{S}_2$ if the meet of $\mathcal{S}$ and $\mathcal{S}_2$ is equal to the meet of $\mathcal{S}_1$ and $\mathcal{S}_2$;

- $\mathcal{S}$ is a *powerset simplification* with respect to $\mathcal{S}_1$ if $q \leq m$.

- $\mathcal{S}$ is a *disjunct meet-preserving simplification* with respect to $\mathcal{S}_1$ if, for each $s_k \in \mathcal{S}$, there exists $d_i \in \mathcal{S}_1$ such that, for each $c_j \in \mathcal{S}_2$, s_k is a meet-preserving enlargement and simplification of d_i using context c_j.

The library provides a binary operator (`simplify_using_context`) for the pointset powerset domain that returns a powerset which is a powerset meet-preserving, powerset simplification and disjunct meet-preserving simplification of its first argument using the second argument as context.

Notice that, due to the powerset simplification property, in general a meet-preserving powerset simplification is *not* an enlargement with respect to the ordering defined on the powerset lattice. Because of this, the operator provided by the library is only well-defined when the base-level domain is not itself a powerset domain.

1.13.2 Geometric Comparisons

Given the pointset powersets $\mathcal{S}_1, \mathcal{S}_2$ over the same base-level domain and with the same space dimension, then we say that $\mathcal{S}_1$ *geometrically covers* $\mathcal{S}_2$ if every point (in some disjunct) of $\mathcal{S}_2$ is also a point in a disjunct of $\mathcal{S}_1$. If $\mathcal{S}_1$ geometrically covers $\mathcal{S}_2$ and $\mathcal{S}_2$ geometrically covers $\mathcal{S}_1$, then we say that they are *geometrically equal*.

1.13.3 Pairwise Merge

Given the pointset powerset $\mathcal{S}$ over a base-level semantic GD domain D, then the *pairwise merge* operator takes pairs of distinct elements in $\mathcal{S}$ whose upper bound (denoted here by $\uplus$) in D (using the PPL operator `upper_bound_assign()` for D) is the same as their set-theoretical union and replaces them by their union. This replacement is done recursively so that, for each pair c, d of distinct disjuncts in the result set, we have $c \uplus d \neq c \cup d$.

1.13.4 Powerset Extrapolation Operators

The library implements a generalization of the extrapolation operator for powerset domains proposed in [BGP99]. The operator `BGP99_extrapolation_assign` is made parametric by allowing for the specification of any PPL extrapolation operator for the base-level domain. Note that, even when the extrapolation operator for the base-level domain D is known to be a widening on D, the `BGP99_extrapolation_assign` operator cannot guarantee the convergence of the iteration sequence in a finite number of steps (for a counter-example, see [BHZ04]).

1.13.5 Certificate-Based Widenings

The PPL library provides support for the specification of proper widening operators on the pointset powerset domain. In particular, this version of the library implements an instance of the *certificate-based widening framework* proposed in [BHZ03b].

A *finite convergence certificate* for an extrapolation operator is a formal way of ensuring that such an operator is indeed a widening on the considered domain. Given a widening operator on the base-level domain D, together with the corresponding convergence certificate, the BHZ03 framework is able to lift this widening on D to a widening on the pointset powerset domain; ensuring convergence in a finite number of iterations.

Being highly parametric, the BHZ03 widening framework can be instantiated in many ways. The current implementation provides the templatic operator `BHZ03_widening_assign<Certificate, Widening>` which only exploits a fraction of this generality, by allowing the user to specify the base-level widening function and the corresponding certificate. The widening strategy is fixed and uses two extrapolation heuristics: first, the upper bound operator for the base-level domain is tried; second, the BGP99 extrapolation operator is tried, possibly applying pairwise merging. If both heuristics fail to converge according to the convergence certificate, then an attempt is made to apply the base-level widening to the upper bound of the two arguments, possibly improving the result obtained by means of the difference operator for the base-level domain. For more details and a justification of the overall approach, see [BHZ03b] and [BHZ04].

The library provides several convergence certificates. Note that, for the domain of Polyhedra, while Parma_Polyhedra_Library::BHRZ03_Certificate the "BHRZ03_Certificate" is compatible with both the BHRZ03 and the H79 widenings, H79_Certificate is only compatible with the latter. Note that using different certificates will change the results obtained, even when using the same base-level widening operator. It is also worth stressing that it is up to the user to see that the widening operator is actually compatible

with a given convergence certificate. If such a requirement is not met, then an extrapolation operator will be obtained.

1.14 Using the Library

1.14.1 A Note on the Implementation of the Operators

When adopting the double description method for the representation of convex polyhedra, the implementation of most of the operators may require an explicit conversion from one of the two representations into the other one, leading to algorithms having a worst-case exponential complexity. However, thanks to the adoption of lazy and incremental computation techniques, the library turns out to be rather efficient in many practical cases.

In earlier versions of the library, a number of operators were introduced in two flavors: a *lazy* version and an *eager* version, the latter having the operator name ending with `_and_minimize`. In principle, only the lazy versions should be used. The eager versions were added to help a knowledgeable user obtain better performance in particular cases. Basically, by invoking the eager version of an operator, the user is trading laziness to better exploit the incrementality of the inner library computations. Starting from version 0.5, the lazy and incremental computation techniques have been refined to achieve a better integration: as a consequence, the lazy versions of the operators are now almost always more efficient than the eager versions.

One of the cases when an eager computation might still make sense is when the well-known *fail-first* principle comes into play. For instance, if you have to compute the intersection of several polyhedra and you strongly suspect that the result will become empty after a few of these intersections, then you may obtain a better performance by calling the eager version of the intersection operator, since the minimization process also enforces an emptiness check. Note anyway that the same effect can be obtained by interleaving the calls of the lazy operator with explicit emptiness checks.

Warning

> For the reasons mentioned above, starting from version 0.10 of the library, the usage of the eager versions (i.e., the ones having a name ending with `_and_minimize`) of these operators is *deprecated*; this is in preparation of their complete removal, which will occur starting from version 0.11.

1.14.2 On Pointset_Powerset and Partially_Reduced_Product Domains: A Warning

For future versions of the PPL library all practical instantiations for the disjuncts for a pointset_powerset and component domains for the partially_reduced_product domains will be fully supported. However, for version 0.10, these compound domains should not themselves occur as one of their argument domains. Therefore their use comes with the following warning.

Warning

> The `Pointset_Powerset<PSET>` and `Partially_Reduced_Product<D1, D2, R>` should only be used with the following instantiations for the disjunct domain template `PSET` and component domain templates `D1` and `D2`: `C_Polyhedron`, `NNC_Polyhedron`, `Grid`, `Octagonal_Shape<T>`, `BD_Shape<T>`, `Box<T>`.

1.14.3 On Object-Orientation and Polymorphism: A Disclaimer

The PPL library is mainly a collection of so-called "concrete data types": while providing the user with a clean and friendly interface, these types are not meant to --- i.e., they should not --- be used polymorphically

(since, e.g., most of the destructors are not declared `virtual`). In practice, this restriction means that the library types should not be used as *public base classes* to be derived from. A user willing to extend the library types, adding new functionalities, often can do so by using *containment* instead of inheritance; even when there is the need to override a `protected` method, non-public inheritance should suffice.

1.14.4 On Const-Correctness: A Warning about the Use of References and Iterators

Most operators of the library depend on one or more parameters that are declared "const", meaning that they will not be changed by the application of the considered operator. Due to the adoption of lazy computation techniques, in many cases such a const-correctness guarantee only holds at the semantic level, whereas it does not necessarily hold at the implementation level. For a typical example, consider the extraction from a polyhedron of its constraint system representation. While this operation is not going to change the polyhedron, it might actually invoke the internal conversion algorithm and modify the generators representation of the polyhedron object, e.g., by reordering the generators and removing those that are detected as redundant. Thus, any previously computed reference to the generators of the polyhedron (be it a direct reference object or an indirect one, such as an iterator) will no longer be valid. For this reason, code fragments such as the following should be avoided, as they may result in undefined behavior:

```
// Find a reference to the first point of the non-empty polyhedron 'ph'.
const Generator_System& gs = ph.generators();
Generator_System::const_iterator i = gs.begin();
for (Generator_System::const_iterator gs_end = gs.end(); i != gs_end; ++i)
  if (i->is_point())
    break;
const Generator& p = *i;
// Get the constraints of 'ph'.
const Constraint_System& cs = ph.constraints();
// Both the const iterator 'i' and the reference 'p'
// are no longer valid at this point.
cout << p.divisor() << endl;  // Undefined behavior!
++i;                          // Undefined behavior!
```

As a rule of thumb, if a polyhedron plays any role in a computation (even as a const parameter), then any previously computed reference to parts of the polyhedron may have been invalidated. Note that, in the example above, the computation of the constraint system could have been placed after the uses of the iterator `i` and the reference `p`. Anyway, if really needed, it is always possible to take a copy of, instead of a reference to, the parts of interest of the polyhedron; in the case above, one may have taken a copy of the generator system by replacing the second line of code with the following:

```
Generator_System gs = ph.generators();
```

The same observations, modulo syntactic sugar, apply to the operators defined in the C interface of the library.

1.15 Bibliography

[Anc91] C. Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales*. PhD thesis, Université de Paris VI, Paris, France, March 1991.

[BA05] J. M. Bjorndalen and O. Anshus. Lessons learned in benchmarking - Floating point benchmarks: Can you trust them? In *Proceedings of the Norsk informatikkonferanse 2005 (NIK 2005)*, pages 89-100, Bergen, Norway, 2005. Tapir Akademisk Forlag.

[Bag97] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, March 1997. Printed as Report TD-1/97.

[Bag98] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. *Science of Computer Programming*, 30(1-2):119-155, 1998.

[BCC+ 02] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In T. Æ. Mogensen, D. A. Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*, volume 2566 of *Lecture Notes in Computer Science*, pages 85-108. Springer-Verlag, Berlin, 2002.

[BDH+ 05] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. A linear domain for analyzing the distribution of numerical values. Report 2005.06, School of Computing, University of Leeds, UK, 2005. Available at http://www.comp.leeds.ac.uk/research/pubs/reports.shtml.

[BDH+ 06] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. A practical tool for analyzing the distribution of numerical values, 2006. Available at http://www.comp.leeds.ac.uk/hill/Papers/papers.html.

[BDH+ 07] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. Grids: A domain for analyzing the distribution of numerical values. In G. Puebla, editor, *Logic-based Program Synthesis and Transformation, 16th International Symposium*, volume 4407 of *Lecture Notes in Computer Science*, pages 219-235, Venice, Italy, 2007. Springer-Verlag, Berlin.

[BFT00] A. Bemporad, K. Fukuda, and F. D. Torrisi. Convexity recognition of the union of polyhedra. Report AUT00-13, Automatic Control Laboratory, ETHZ, Zurich, Switzerland, 2000.

[BFT01] A. Bemporad, K. Fukuda, and F. D. Torrisi. Convexity recognition of the union of polyhedra. *Computational Geometry: Theory and Applications*, 18(3):141-154, 2001.

[BGP99] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747-789, 1999.

[BHMZ04] R. Bagnara, P. M. Hill, E. Mazzi, and E. Zaffanella. Widening operators for weakly-relational numeric abstractions. Report arXiv:cs.PL/0412043, 2004. Extended abstract. Contribution to the *International workshop on "Numerical & Symbolic Abstract Domains"* (NSAD'05, Paris, January 21, 2005). Available at http://arxiv.org/ and http://www.cs.unipr.it/ppl/.

[BHMZ05a] R. Bagnara, P. M. Hill, E. Mazzi, and E. Zaffanella. Widening operators for weakly-relational numeric abstractions. Quaderno 399, Dipartimento di Matematica, Università di Parma, Italy, 2005. Available at http://www.cs.unipr.it/Publications/.

[BHMZ05b] R. Bagnara, P. M. Hill, E. Mazzi, and E. Zaffanella. Widening operators for weakly-relational numeric abstractions. In C. Hankin and I. Siveroni, editors, *Static Analysis: Proceedings of the 12th International Symposium*, volume 3672 of *Lecture Notes in Computer Science*, pages 3-18, London, UK, 2005. Springer-Verlag, Berlin.

[BHRZ03a] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337-354, San Diego, California, USA, 2003. Springer-Verlag, Berlin.

[BHRZ03b] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. Quaderno 312, Dipartimento di Matematica, Università di Parma, Italy, 2003. Available at http://www.cs.unipr.it/Publications/.

[BHRZ05] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. *Science of Computer Programming*, 58(1-2):28-56, 2005.

[BHZ02a] R. Bagnara, P. M. Hill, and E. Zaffanella. A new encoding and implementation of not necessarily closed convex polyhedra. Quaderno 305, Dipartimento di Matematica, Università di Parma, Italy, 2002. Available at http://www.cs.unipr.it/Publications/.

[BHZ02b] R. Bagnara, P. M. Hill, and E. Zaffanella. A new encoding of not necessarily closed convex polyhedra. In M. Carro, C. Vacheret, and K.-K. Lau, editors, *Proceedings of the 1st CoLogNet Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems*, pages 147-153, Madrid, Spain, 2002. Published as TR Number CLIP4/02.0, Universidad Politécnica de Madrid, Facultad de Informática.

[BHZ03a] R. Bagnara, P. M. Hill, and E. Zaffanella. A new encoding and implementation of not necessarily closed convex polyhedra. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proceedings of the 3rd Workshop on Automated Verification of Critical Systems*, pages 161-176, Southampton, UK, 2003. Published as TR Number DSSE-TR-2003-2, University of Southampton.

[BHZ03b] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. In B. Steffen and G. Levi, editors, *Verification, Model Checking and Abstract Interpretation: Proceedings of the 5th International Conference (VMCAI 2004)*, volume 2937 of *Lecture Notes in Computer Science*, pages 135-148, Venice, Italy, 2003. Springer-Verlag, Berlin.

[BHZ04] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. Quaderno 349, Dipartimento di Matematica, Università di Parma, Italy, 2004. Available at http://www.cs.unipr.it/Publications/.

[BHZ05] R. Bagnara, P. M. Hill, and E. Zaffanella. Not necessarily closed convex polyhedra and the double description method. *Formal Aspects of Computing*, 17(2):222-257, 2005.

[BHZ06a] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Quaderno 457, Dipartimento di Matematica, Università di Parma, Italy, 2006. Available at http://www.cs.unipr.it/Publications/. Also published as arXiv:cs.MS/0612085, available from http://arxiv.org/.

[BHZ06b] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *Software Tools for Technology Transfer*, 8(4/5):449-466, 2006. In the printed version of this article, all the figures have been improperly printed (rendering them useless). See [BHZ07c].

[BHZ07a] R. Bagnara, P. M. Hill, and E. Zaffanella. Applications of polyhedral computations to the analysis and verification of hardware and software systems. Quaderno 458, Dipartimento di Matematica, Università di Parma, Italy, 2007. Available at http://www.cs.unipr.it/Publications/. Also published as arXiv:cs.CG/0701122, available from http://arxiv.org/.

[BHZ07b] R. Bagnara, P. M. Hill, and E. Zaffanella. An improved tight closure algorithm for integer octagonal constraints. Quaderno 467, Dipartimento di Matematica, Università di Parma, Italy, 2007. Available at http://www.cs.unipr.it/Publications/. Also published as arXiv:0705.4618v2 [cs.DS], available from http://arxiv.org/.

[BHZ07c] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *Software Tools for Technology Transfer*, 9(3/4):413-414, 2007. Erratum to [BHZ06b] containing all the figures properly printed.

[BHZ08a] R. Bagnara, P. M. Hill, and E. Zaffanella. An improved tight closure algorithm for integer octagonal constraints. In F. Logozzo, D. Peled, and L. Zuck, editors, *Verification, Model Checking and Abstract Interpretation: Proceedings of the 9th International Conference (VMCAI 2008)*, volume 4905 of *Lecture Notes in Computer Science*, pages 8-21, San Francisco, USA, 2008. Springer-Verlag, Berlin.

[BHZ08b] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1-2):3-21, 2008.

[BHZ09a] R. Bagnara, P. M. Hill, and E. Zaffanella. Applications of polyhedral computations to the analysis and verification of hardware and software systems. *Theoretical Computer Science*, 410(46):4672-4691, 2009.

[BHZ09b] R. Bagnara, P. M. Hill, and E. Zaffanella. Exact join detection for convex polyhedra and other numerical abstractions. Quaderno 492, Dipartimento di Matematica, Università di Parma, Italy, 2009. Available at http://www.cs.unipr.it/Publications/. A corrected and improved version (corrected an error in the statement of condition (3) of Theorem 3.6, typos corrected in statement and proof of Theorem 6.8) has been published in [BHZ09c].

[BHZ09c] R. Bagnara, P. M. Hill, and E. Zaffanella. Exact join detection for convex polyhedra and other numerical abstractions. Report arXiv:cs.CG/0904.1783, 2009. Available at http://arxiv.org/ and http://www.cs.unipr.it/ppl/.

[BHZ09d] R. Bagnara, P. M. Hill, and E. Zaffanella. Weakly-relational shapes for numeric abstractions: Improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279-323, 2009.

[BHZ10] R. Bagnara, P. M. Hill, and E. Zaffanella. Exact join detection for convex polyhedra and other numerical abstractions. *Computational Geometry: Theory and Applications*, 43(5):453-473, 2010. To appear in print. Available online at http://dx.doi.org/10.1016/j.comgeo.2009.09.002.

[BJT99] F. Besson, T. P. Jensen, and J.-P. Talpin. Polyhedral analysis for synchronous languages. In A. Cortesi and G. Filé, editors, *Static Analysis: Proceedings of the 6th International Symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 51-68, Venice, Italy, 1999. Springer-Verlag, Berlin.

[BK89] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In B. Knobe, editor, *Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation (PLDI)*, volume 24(7) of *ACM SIGPLAN Notices*, pages 41-53, Portland, Oregon, USA, 1989. ACM Press.

[BMPZ10] R. Bagnara, F. Mesnard, A. Pescetti, and E. Zaffanella. The automatic synthesis of linear ranking functions: The complete unabridged version. Quaderno 498, Dipartimento di Matematica, Università di Parma, Italy, 2010. Available at http://www.cs.unipr.it/Publications/. Also published as arXiv:cs.PL/1004.0944, available from http://arxiv.org/.

[BRZH02a] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 213-229, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[BRZH02b] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. Quaderno 286, Dipartimento di Matematica, Università di Parma, Italy, 2002. See also [BRZH02c]. Available at http://www.cs.unipr.it/Publications/.

[BRZH02c] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Errata for technical report "Quaderno 286". Available at http://www.cs.unipr.it/Publications/, 2002. See [BRZH02b].

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In B. Robinet, editor, *Proceedings of the Second International Symposium on Programming*, pages 106-130, Paris, France, 1976. Dunod, Paris, France.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269-282, New York, 1979. ACM Press.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269-295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84-96, Tucson, Arizona, 1978. ACM Press.

[Che64] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear equations. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 4(4):151-158, 1964.

[Che65] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5(2):228-233, 1965.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282-293, 1968.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[FCB07] P. Feautrier, J.-F. Collard, and C. Bastoul. *PIP/PipLib: A Solver for Parametric Integer Programming Problems*, 5.0 edition, July 2007. Distributed with PIP/PipLib 1.4.0.

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22(3):243-268, 1988.

[FP96] K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and Y. Manoussakis, editors, *Combinatorics and Computer Science, 8th Franco-Japanese and 4th Franco-Chinese Conference, Brest, France, July 3-5, 1995, Selected Papers*, volume 1120 of *Lecture Notes in Computer Science*, pages 91-111. Springer-Verlag, Berlin, 1996.

[Fuk98] K. Fukuda. Polyhedral computation FAQ. Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland, available at `http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html`, 1998.

[GDD+ 04] D. Gopan, F. DiMaio, N. Dor, T. W. Reps, and M. Sagiv. Numeric domains with summarized dimensions. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 512-529, Barcelona, Spain, 2004. Springer-Verlag, Berlin.

[GJ00] E. Gawrilow and M. Joswig. `polymake`: A framework for analyzing convex polytopes. In G. Kalai and G. M. Ziegler, editors, *Polytopes - Combinatorics and Computation*, pages 43-74. Birkhäuser, 2000.

[GJ01] E. Gawrilow and M. Joswig. `polymake`: An approach to modular software design in computational geometry. In *Proceedings of the 17th Annual Symposium on Computational Geometry*, pages 222-231, Medford, MA, USA, 2001. ACM.

[GR77] D. Goldfarb and J. K. Reid. A practical steepest-edge simplex algorithm. *Mathematical Proramming*, 12(1):361-371, 1977.

[Gra91] P. Granger. Static analysis of linear congruence equalities among variables of a program. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP'91)*, volume 493 of *Lecture Notes in Computer Science*, pages 169-192, Brighton, UK, 1991. Springer-Verlag, Berlin.

[Gra97] P. Granger. Static analyses of congruence properties on rational numbers (extended abstract). In P. Van Hentenryck, editor, *Static Analysis: Proceedings of the 4th International Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 278-292, Paris, France, 1997. Springer-Verlag, Berlin.

[Hal79] N. Halbwachs. *Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme*. Thèse de 3ème cycle d'informatique, Université scientifique et médicale de Grenoble, Grenoble, France, March 1979.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *Computer Aided Verification: Proceedings of the 5th International Conference (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 333-346, Elounda, Greece, 1993. Springer-Verlag, Berlin.

[HH95] T. A. Henzinger and P.-H. Ho. A note on abstract interpretation strategies for hybrid automata. In P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 252-264. Springer-Verlag, Berlin, 1995.

[HHL90] L. Huelsbergen, D. Hahn, and J. Larus. Exact dependence analysis using data access descriptors. Technical Report 945, Department of Computer Science, University of Wisconsin, Madison, 1990.

[HKP95] N. Halbwachs, A. Kerbrat, and Y.-E. Proy. *POLyhedra INtegrated Environment*. Verimag, France, version 1.0 of POLINE edition, September 1995. Documentation taken from source code.

[HLW94] V. Van Dongen H. Le Verge and D. K. Wilde. Loop nest synthesis using the polyhedral library. *Publication interne* 830, IRISA, Campus de Beaulieu, Rennes, France, 1994.

[HMT71] L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras: Part I*. North-Holland, Amsterdam, 1971.

[HPR94] N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In B. Le Charlier, editor, *Static Analysis: Proceedings of the 1st International Symposium*, volume 864 of *Lecture Notes in Computer Science*, pages 223-237, Namur, Belgium, 1994. Springer-Verlag, Berlin.

[HPR97] N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157-185, 1997.

[HPWT01] T. A. Henzinger, J. Preussig, and H. Wong-Toi. Some lessons from the hytech experience. In *Proceedings of the 40th Annual Conference on Decision and Control*, pages 2887-2892. IEEE Computer Society Press, 2001.

[Jea02] B. Jeannet. *Convex Polyhedra Library*, release 1.1.3c edition, March 2002. Documentation of the "New Polka" library available at http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html.

[JMSY94] J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond finite domains. In A. Borning, editor, *Principles and Practice of Constraint Programming: Proceedings of the Second International Workshop*, volume 874 of *Lecture Notes in Computer Science*, pages 86-94, Rosario, Orcas Island, Washington, USA, 1994. Springer-Verlag, Berlin.

[KBB+ 06] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. *Discrete and Computational Geometry*, 2006. Invited contribution. To appear.

[Kuh56] H. W. Kuhn. Solvability and consistency for linear equations and inequalities. *American Mathematical Monthly*, 63:217-232, 1956.

[Le 92] 92 H. Le Verge. A note on Chernikova's algorithm. *Publication interne* 635, IRISA, Campus de Beaulieu, Rennes, France, 1992.

[Loe99] V. Loechner. *PolyLib*: A library for manipulating parameterized polyhedra. Available at http://icps.u-strasbg.fr/~loechner/polylib/, March 1999. Declares itself to be a continuation of [Wil93].

[LW97] V. Loechner and D. K. Wilde. Parameterized polyhedra and their vertices. *International Journal of Parallel Programming*, 25(6):525-549, 1997.

[Mas92] F. Masdupuy. Array operations abstraction using semantic analysis of trapezoid congruences. In *Proceedings of the 6th ACM International Conference on Supercomputing*, pages 226-235, Washington, DC, USA, 1992. ACM Press.

[Mas93] F. Masdupuy. *Array Indices Relational Semantic Analysis Using Rational Cosets and Trapezoids*. Thèse d'informatique, École Polytechnique, Palaiseau, France, December 1993.

[Min01a] A. Miné. A new numerical abstract domain based on difference-bound matrices. In O. Danvy and A. Filinski, editors, *Proceedings of the 2nd Symposium on Programs as Data Objects (PADO 2001)*, volume 2053 of *Lecture Notes in Computer Science*, pages 155-172, Aarhus, Denmark, 2001. Springer-Verlag, Berlin.

[Min01b] A. Miné. The octagon abstract domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, pages 310-319, Stuttgart, Germany, 2001. IEEE Computer Society Press.

[Min02] A. Miné. A few graph-based relational numerical abstract domains. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 117-132, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[Min04] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. Schmidt, editor, *Programming Languages and Systems: Proceedings of the 13th European Symposium on Programming*, volume 2986 of *Lecture Notes in Computer Science*, pages 3-17, Barcelona, Spain, 2004. Springer-Verlag, Berlin.

[Min05] A. Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Paris, France, March 2005.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games - Volume II*, number 28 in Annals of Mathematics Studies, pages 51-73. Princeton University Press, Princeton, New Jersey, 1953.

[NF01] T. Nakanishi and A. Fukuda. Modulo interval arithmetic and its application to program analysis. *Transactions of Information Processing Society of Japan*, 42(4):829-837, 2001.

[NJPF99] T. Nakanishi, K. Joe, C. D. Polychronopoulos, and A. Fukuda. The modulo interval: A simple and practical representation for program analysis. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pages 91-96, Newport Beach, California, USA, 1999. IEEE Computer Society.

[NO77] G. Nelson and D. C. Oppen. Fast decision algorithms based on Union and Find. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 114-119, Providence, RI, USA, 1977. IEEE Computer Society Press. The journal version of this paper is [NO80].

[NO80] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356-364, 1980. An earlier version of this paper is [NO77].

[NR00] S. P. K. Nookala and T. Risset. A library for Z-polyhedral operations. *Publication interne* 1330, IRISA, Campus de Beaulieu, Rennes, France, 2000.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.

[Pra77] V. R. Pratt. Two easy theories whose combination is hard. Memo sent to Nelson and Oppen concerning a preprint of their paper [NO77], September 1977.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, second edition, 1998.

[QRR96] P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra. Technical Report 1016, IRISA, Campus Universitaire de Bealieu, Rennes, France, July 1996.

[QRR97] P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra using a canonic representation. *Parallel Processing Letters*, 7(2):181-194, 1997.

[QRW00] F. Quilleré, S. V. Rajopadhye, and D. Wilde. Generation of efficient nested loops from polyhedra. *International Journal of Parallel Programming*, 28(5):469-498, 2000.

[RBL06] T. W. Reps, G. Balakrishnan, and J. Lim. Intermediate-representation recovery from low-level code. In J. Hatcliff and F. Tip, editors, *Proceedings of the 2006 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, pages 100-111, Charleston, South Carolina, USA, 2006. ACM Press.

[Ric02] E. Ricci. Rappresentazione e manipolazione di poliedri convessi per l'analisi e la verifica di programmi. Laurea dissertation, University of Parma, Parma, Italy, July 2002. In Italian.

[Sch99] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1999.

[Sho81] R. E. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769-779, 1981.

[SK07] A. Simon and A. King. Taming the wrapping of integer arithmetic. In H. Riis Nielson and G. Filé, editors, *Static Analysis: Proceedings of the 14th International Symposium*, volume 4634 of *Lecture Notes in Computer Science*, pages 121-136, Kongens Lyngby, Denmark, 2007. Springer-Verlag, Berlin.

[Sri93] D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3-4):315-343, 1993.

[SS07a] R. Sen and Y. N. Srikant. Executable analysis using abstract interpretation with circular linear progressions. In *Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2007)*, pages 39-48, Nice, France, 2007. IEEE Computer Society Press.

[SS07b] R. Sen and Y. N. Srikant. Executable analysis with circular linear progressions. Technical Report IISc-CSA-TR-2007-3, Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India, 2007.

[SW70] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I*. Springer-Verlag, Berlin, 1970.

[War03] H. S. Warren, Jr. *Hacker's Delight*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Wey35] H. Weyl. Elementare theorie der konvexen polyeder. *Commentarii Mathematici Helvetici*, 7:290-306, 1935. English translation in [Wey50].

[Wey50] H. Weyl. The elementary theory of convex polyhedra. In H. W. Kuhn, editor, *Contributions to the Theory of Games - Volume I*, number 24 in Annals of Mathematics Studies, pages 3-18. Princeton University Press, Princeton, New Jersey, 1950. Translated from [Wey35] by H. W. Kuhn.

[Wil93] D. K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, December 1993. Also published as IRISA *Publication interne* 785, Rennes, France, 1993.

1.16 Further Notation and Terminology

1.16.1 Linear Independence

A finite set of points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} \subseteq \mathbb{R}^n$ is *linearly independent* if, for all $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the set of equations

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i = \boldsymbol{0}$$

implies that, for each $i = 1, \ldots, k$, $\lambda_i = 0$.

The maximum number of linearly independent points in $\mathbb{R}^n$ is n. Note that linear independence implies affine independence, but the converse is not true.

Proposition

If A is an $m \times n$ matrix, the maximum number of linearly independent rows of A, viewed as vectors of $\mathbb{R}^n$, equals the maximum number of linearly independent columns of A, viewed as vectors of $\mathbb{R}^m$.

1.16.2 Rank

The maximum number of linearly independent rows (columns) of a matrix A is the *rank* of A and is denoted by $\mathrm{rank}(A)$.

Proposition

A polyhedron is a convex set.

1.16.3 Minkowski's Theorem

Let $\mathcal{P} = \{ \boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \geq \boldsymbol{b} \}$ be a non-empty polyhedron where $\mathrm{rank}(A) = n$. Let V be the set of vertices and R the set of extreme rays of $\mathcal{P}$. Let also $\mathcal{V}$ be the set of convex combinations of V and $\mathcal{R}$ the set of positive combinations of R. Then

$$\mathcal{P} = \mathcal{V} + \mathcal{R}.$$

Informally, this theorem states that, whenever a polyhedron $\mathcal{P}$ has a vertex, there exists a decomposition such that

- V is the set of all *vertices* of $\mathcal{P}$;
- R is the set of all *extreme* rays of $\mathcal{P}$; and
- $L = \varnothing$.

The conditions that $\mathcal{P}$ is not empty and $\mathrm{rank}(A) = n$ are equivalent to the condition that $\mathcal{P}$ has a vertex. (See also Nemhauser and Wolsey - Integer and Combinatorial Optimization - propositions 4.1 and 4.2 on pages 92 and 93).

Proposition

Under the same hypotheses of Minkowski's theorem, if $\mathcal{P}$ is a rational polyhedron then all the vertices in V have rational coefficients and we can consider a set R of extreme rays having rational coefficients only.

The second theorem, called Weyl's theorem, states that any system of generators having rational coefficients defines a rational polyhedron:

1.16.4 Weyl's Theorem

If A is a rational $m \times n$ matrix, B is a rational $m' \times n$ matrix and

$$\mathcal{Q} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{x}^{\mathrm{T}} = \boldsymbol{y}^{\mathrm{T}} A + \boldsymbol{z}^{\mathrm{T}} B, \\ \boldsymbol{y} = (y_0, \ldots, y_{m-1})^{\mathrm{T}} \in \mathbb{R}^m_+, \sum_{k=0}^{m-1} y_k = 1, \\ \boldsymbol{z} \in \mathbb{R}^{m'}_+ \end{array} \right\},$$

then $\mathcal{Q}$ is a rational polyhedron.

In fact, since $\mathcal{Q}$ consists of the sum of convex combinations of the rows of A with positive combinations of the rows of B, we can think of A as the matrix of vertices and B as the matrix of rays.

1.16.5 Cone

A set $C \subseteq \mathbb{R}^n$ is a *cone* if

$$\boldsymbol{x} \in C \Rightarrow \lambda \boldsymbol{x} \in C \text{ for all } \lambda \in \mathbb{R}_+.$$

1.16.6 Polyhedral Cone

The polyhedron $\mathcal{P} = \{ \boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \geq \boldsymbol{0} \}$ is a convex cone and is called *polyhedral cone*.

A polyhedral cone is either *pointed*, having the origin as its only vertex, or has no vertices at all.

1.16.7 Lineality Space

Given a polyhedron $\mathcal{P} = \{ \boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \geq \boldsymbol{b} \}$, the *lineality space* of $\mathcal{P}$ is the set

$$\{ \boldsymbol{x} \in \mathcal{P} \mid A\boldsymbol{x} = \boldsymbol{0} \}$$

and it is denoted by $\mathrm{lin.space}(\mathcal{P})$.

1.17 Homogeneous Systems

To simplify the operations on polyhedra, each polyhedron is first transformed to a homogeneous cone in which the original polyhedron is embedded.

1.17.1 Corresponding Polyhedral Cone

The transformation changes the inhomogeneous system of constraints in n variables, representing a polyhedron $\mathcal{P} \in \mathbb{R}^n$, into a homogeneous system in $n+1$ variables, representing a polyhedral cone $C \in \mathbb{R}^{n+1}$, so that each point $\boldsymbol{x} \in \mathcal{P}$ corresponds to a point $\boldsymbol{x}' = (\xi \boldsymbol{x}^{\mathrm{T}}, \xi)^{\mathrm{T}} \in C$ where $\xi \geq 0$. That is,

$$\mathcal{P} = \{ \boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{b} \} = \{ \boldsymbol{x} \mid A\boldsymbol{x} - \boldsymbol{b} \geq \boldsymbol{0} \}$$

$$C = \{ (\xi \boldsymbol{x}^{\mathrm{T}}, \xi)^{\mathrm{T}} \mid \xi A\boldsymbol{x} - \xi \boldsymbol{b} \geq \boldsymbol{0}, \xi \geq 0 \} = \{ \boldsymbol{x}' \mid A'\boldsymbol{x}' \geq \boldsymbol{0} \}$$

where: $\boldsymbol{x}' = (\xi \boldsymbol{x}^{\mathrm{T}}, \xi)^{\mathrm{T}} \in \mathbb{R}^{n+1}$; A' is the $(m+1) \times (n+1)$ matrix having, for its first m rows, the submatrix $(A, -\boldsymbol{b}) \in \mathbb{R}^m \times \mathbb{R}^{n+1}$; and, for the $(m+1)$'st row, $(\boldsymbol{0}^{\mathrm{T}}, 1)$ where $\boldsymbol{0} \in \mathbb{R}^n$. We call C the *corresponding polyhedral cone for* $\mathcal{P}$.

The $(m+1)$'st row $(\boldsymbol{0}^{\mathrm{T}}, 1)$ represents the *positivity constraint* $1 \geq 0$.

Note that $\mathcal{P}$ is contained in C since the intersection of C with the hyperplane defined by the equality $\xi = 1$ is $\mathcal{P}$. Therefore, it is always possible to transform a polyhedron $\mathcal{P}$ to its corresponding polyhedral cone C and then recover $\mathcal{P}$ by means of this intersection.

As C always includes the origin and, hence, is non-empty, by Minkowski's theorem, it can also be represented by a system of generators.

The systems of generators for $\mathcal{P}$ and C are such that:

- Each vertex $\boldsymbol{v}$ in $\mathcal{P}$ corresponds to a ray $(\boldsymbol{v}^{\mathrm{T}}, d)^{\mathrm{T}}$ with $d \neq 0$, in C.
- Each ray $\boldsymbol{r}$ in $\mathcal{P}$ corresponds to the ray $(\boldsymbol{r}^{\mathrm{T}}, 0)^{\mathrm{T}}$ in C.
- Every ray in C corresponds to a vertex or ray in $\mathcal{P}$.
- The origin in $\mathbb{R}^{n+1}$ is a point in C.

Thus, in the cone C, a ray derived from a vertex in $\mathcal{P}$ differs from a ray derived from a ray in $\mathcal{P}$ only in that, for a vertex, the $(n+1)$'st term is different from zero and, for a ray, it is zero.

1.17.2 Double Description

Let $\mathcal{P} \in \mathbb{R}^n$ be a polyhedron and $C \in \mathbb{R}^{n+1}$ the corresponding polyhedral cone. Then the dual representations, the systems of constraints and generators representing C, form the *double description* for $\mathcal{P}$.

Note that, in a double description for a non-empty polyhedron, the system of constraints subsumes the positivity constraint $1 \geq 0$ while the system of generators (which has only rays and lines corresponding to the vertices, rays and lines for $\mathcal{P}$) implicitly assumes the origin in $\mathbb{R}^{n+1}$ as a point so that the cone C represented by the generators is non-empty.

1.17.3 PPL Polyhedron Representation

In the PPL, a polyhedron is represented by one or both of the representations in its double description. Thus, in the sequel, by *PPL representation of a polyhedra*, we are referring to the corresponding representation of its corresponding polyhedral cone.

1.17.4 Valid Linear Inequalities

Let $\mathcal{P}$ be a convex polyhedron (or polytope) in $\mathbb{R}^n$. For a real n-vector $\boldsymbol{c}$ and a real number b, a linear inequality $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \geq b$ (briefly denoted by $(\boldsymbol{c}, b)$) is called *valid* for $\mathcal{P}$ if it is satisfied by all points $\boldsymbol{x} \in \mathcal{P}$.

1.17.5 Redundancy

1. In a system of equalities, if an equality is a linear combination of the others, it is said to be *dependent* upon them; the dependent equality is called *redundant*. A system containing no redundant equality is called *independent*.

2. In a system of inequalities, an inequality is said to be *redundant* if it can be eliminated from the system obtaining a system equivalent to the previous one, i.e., having the same solutions.

Given a polyhedron $\mathcal{P}$ generated by V vertices, R rays and L lines, we say that:

1. L is *irredundant* if L is a set of linearly independent lines; and

2. a ray $\boldsymbol{r}_1 \in R$ is *redundant* if there exists another ray $\boldsymbol{r}_2 \in R$ and there exists $\lambda \in \mathbb{R}, \lambda > 0$ such that $\boldsymbol{r}_1 = \lambda \boldsymbol{r}_2$.

Note that, in the PPL representation of a polyhedron $\mathcal{P}$, vertices are represented as rays so that this concept of a redundant ray also applies to the vertices of $\mathcal{P}$.

1.17.6 Face

If $(\boldsymbol{c}, b)$ is a valid inequality for $\mathcal{P}$, and $F = \{\, \boldsymbol{x} \in \mathcal{P} \mid \langle \boldsymbol{c}, \boldsymbol{x} \rangle = b \,\}$, F is called a *face* of $\mathcal{P}$ and we say that the inequality represents F. A face F is said to be *proper* if $F \neq \varnothing$ and $F \neq \mathcal{P}$.

When F is non-empty, we say that $(\boldsymbol{c}, b)$ *supports* $\mathcal{P}$.

The empty polyhedron and the universe polyhedron both have no proper faces, because the only face of an empty polyhedron is itself, while the faces of the universe polyhedron are itself and the emptyset.

Let $\mathcal{P}$ be a non-empty polyhedron. The set

$$F = \{\boldsymbol{p}\} + \mathrm{lin.space}(\mathcal{P}),$$

where $\boldsymbol{p}$ is a point of $\mathcal{P}$ and the symbol '$+$' denotes the Minkowski's sum, is a *minimal proper face* of the polyhedron if F is a proper face of $\mathcal{P}$.

1.17.7 Facet

A proper face F of $\mathcal{P}$ is a *facet* (or *maximal proper face*) of $\mathcal{P}$ if it is not strictly included into any other proper face of $\mathcal{P}$. The affine dimension of a facet is equal to $\dim(\mathcal{P}) - 1$.

Proposition

Let $\mathcal{P}$ a polyhedron in $\mathbb{R}^n$. The set of all faces is a lattice under inclusion: the minimal face is the emptyset, while the maximal face is the polyhedron.

Proposition

Let $\mathcal{P} \neq \varnothing$ be a polyhedron in $\mathbb{R}^n$ and C be the polyhedral cone in $\mathbb{R}^{n+1}$ obtained from $\mathcal{P}$ by homogenization, then:

1. the only *minimal proper face* of C is $\mathrm{lin.space}(C)$;

2. let $\boldsymbol{y} \in C$ be different from $\mathbf{0}$ and $\mathrm{cone}\{\boldsymbol{y}\}$ be defined as $\{\, \lambda \boldsymbol{y} \mid \lambda \geq 0 \,\}$. If the set $F = \mathrm{cone}\{\boldsymbol{y}\} + \mathrm{lin.space}(C)$ is a proper face of C, then $\boldsymbol{y}$ is an *extremal ray* of C.

1.17.8 Ray Space

Given the decomposition $\mathcal{V} + \mathcal{R} + \mathcal{L}$ of a polyhedron $\mathcal{P}$ the set $\mathcal{V} + \mathcal{R}$ is called the *ray space* of $\mathcal{P}$ and denoted by $\mathrm{rayspace}(\mathcal{P})$.

Thus a polyhedron $\mathcal{P}$ can be always decomposed in its $\mathrm{lin.space}$ and its $\mathrm{rayspace}$.

Note that, since $\mathrm{lin.space}(\mathcal{P})$ and $\mathrm{rayspace}(\mathcal{P})$ are polyhedra, their affine dimensions can be computed using the definition of affine dimension given for polyhedra.

The spaces defined are connected by some consistency rules shown below.

1.17.9 Dimensionality Rules

In $\mathbb{R}^n$

- The dimension of the $\mathrm{lin.space}$ is the rank of any set of lines that span the space.
- The dimension of the polyhedron is the dimension of the $\mathrm{rayspace}$ plus the dimension of the $\mathrm{lin.space}$.
- The dimension of the $\mathrm{rayspace}$ is n minus the number of irredundant lines minus the number of irredundant equalities.

The proofs of these properties can be obtained considering the definitions of affine dimension and the decomposition of a polyhedron.

1.17.10 Saturation

Let us consider a ray $\boldsymbol{r} \in \mathbb{R}^n$ and an inequality $(\boldsymbol{a}, 0)$ where $\boldsymbol{a} \in \mathbb{R}^n$. Then we say that:

- r *saturates* the inequality if $\langle \boldsymbol{a}, \boldsymbol{r} \rangle = 0$;
- r *verifies* the inequality if $\langle \boldsymbol{a}, \boldsymbol{r} \rangle > 0$;
- r *violates* the inequality if $\langle \boldsymbol{a}, \boldsymbol{r} \rangle < 0$.

Similarly, considering an equality $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = 0$:

- $\boldsymbol{r}$ *saturates* the equality if $\langle \boldsymbol{a}, \boldsymbol{r} \rangle = 0$;
- $\boldsymbol{r}$ *does not verify* the equality if $\langle \boldsymbol{a}, \boldsymbol{r} \rangle \neq 0$.

A constraint (i.e., an equality or an inequality) is *satisfied* by a ray if the ray saturates or verifies the constraint.

Proposition

Let $C \subseteq \mathbb{R}^n$ be a polyhedral cone and $\boldsymbol{y}_1, \boldsymbol{y}_2 \in C$. If the sets $F_i = \mathrm{cone}\{\boldsymbol{y}_i\} + \mathrm{lin.space}(C)$ with $i = 1, 2$ are proper faces of C, F_1 is equal to F_2 if and only if the set of constraints that are saturated by $\boldsymbol{y}_1$ is equal to the set of constraints that are saturated by $\boldsymbol{y}_2$.

1.17.11 Saturation Matrix

A *saturation matrix* is a bit matrix that represents the connection between constraints and generators of a polyhedron. There are two kinds of saturation matrices, one having rows indexed by constraints and columns indexed by generators (*sat_g*), and one (that is the transposed version of the previous one) having rows indexed by generators and columns indexed by constraints (*sat_c*).

For instance, in the saturation matrix sat_g, the elements are defined as follows:

$$s_{ij} = \begin{cases} 0, \text{if the constraint indexed by } i \text{ is saturated by the generator indexed by } j; \\ 1, \text{if the constraint indexed by } i \text{ is only verified by the generator indexed by } j. \end{cases}$$

For efficiency reasons, the PPL uses both the sat_g and sat_c matrices.

1.17.12 Saturation Rule

In an n-dimensional $\mathrm{rayspace}$,

1. Every inequality must be saturated by at least n vertices/rays.
2. Every vertex must saturate at least n inequalities and a ray must saturate at least $n-1$ inequalities plus the positivity constraint.
3. Every equality must be saturated by all lines and vertices/rays.
4. Every line must saturate all equalities and inequalities.

These rules are a consequence of the saturation concept.

Proposition

Let $C = \{\boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{0}\}$ be a polyhedral cone. Then the minimal proper face of C in an n-dimensional space can also be represented as $F = \{\, \boldsymbol{x} \mid A\boldsymbol{x} = \boldsymbol{0} \,\}$.

To see this, note that the minimal proper face of a polyhedral cone is equal to its lineality space. This for definition is composed by all $\boldsymbol{y}$ of C that satisfies $A\boldsymbol{x} = \boldsymbol{0}$.

1.17.13 Adjacent Rays

Let A be representing matrix of constraints of a cone C and Q the set of rays that generate C. Then two rays $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are *adjacent rays* if

1. there exists at least a row of A (i.e., a constraint) that is saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$
2. and none of the rays of Q, except $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, saturates all the constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$.

1.17.14 Independence Rule

1. No inequality is a positive combination of any other two inequalities or equalities.
2. No ray is a linear combination of any other two rays or lines.
3. The set of equalities must be linearly independent.
4. The set of lines must be linearly independent.

To remove redundant constraints/generators we will use the following characterization:

1.17.15 Redundancy Rules

- An inequality is not redundant if it satisfies both point (1) of the saturation rule and point (1) of the independence rule.
- A vertex/ray is irredundant if it satisfies both point (2) of the saturation rule and point (2) of the independence rule.

It is useful to note that:

- All rays saturate the positivity constraint and no vertex saturates the positivity constraint; in fact in the homogeneous form the positivity constraint is represented by the vector $\boldsymbol{a}^{\mathrm{T}} = (0, \ldots, 0, 1)$, rays are of the form $\boldsymbol{r} = (r_0, \ldots, r_{n-1}, 0)^{\mathrm{T}}$ and vertices $\boldsymbol{v} = (v_0, \ldots, v_{n-1}, d)^{\mathrm{T}}$ with $d \neq 0$, thus $\langle \boldsymbol{a}, \boldsymbol{r} \rangle = 0$ for each ray $\boldsymbol{r}$ and $\langle \boldsymbol{a}, \boldsymbol{v} \rangle \neq 0$ for each vertex $\boldsymbol{v}$.
- The positivity constraint will be irredundant if and only if the size of the set of rays is $\geq n$, where n is the dimension of the ray space, and the rank of the ray set is n; in fact a constraint is irredundant if it is saturated by at least n vertices/rays (see above), but since only rays saturate the positivity constraint, then in a system with n vertices/rays the positivity constraint is irredundant.

1.18 Integers Represented by Floating Point Numbers

Floating point numbers can be used to represent finite families of integer numbers. In this section we collect some closure properties of these families that are exploited in the PPL.

In order not to depend on the particular family of floating point numbers considered, we consider an abstraction that is parametric in the number b of bits in the mantissa and gives no limit to the magnitude of the exponent e. For $b \in \mathbb{N} \setminus \{0\}$ let

$$F_b^+ = \left\{ x \in \mathbb{N} \;\middle|\; x = (1 + m/2^b) \cdot 2^e, e \in \mathbb{N}, m \in \mathbb{N} \cap [0, 2^b - 1] \right\},$$
$$F_b = F_b^+ \cup \{0\} \cup \{ -x \mid x \in F_b^+ \}.$$

Let $\phi \colon \mathbb{R} \to \mathbb{Z}$ denote the function defined by

$$\phi(t) = \begin{cases} \lfloor t \rfloor, & \text{if } t \geq 0; \\ \lceil t \rceil, & \text{if } t < 0. \end{cases}$$

Notice that ϕ is an *odd* function, that is, it satisfies $\phi(-t) = -\phi(t)$ for all $t \in \mathbb{R}$. For $x, y \in \mathbb{Z}$ with $y \neq 0$, we also write

$$x \operatorname{div} y = \phi(x/y),$$
$$x \operatorname{rem} y = x - (x \operatorname{div} y) y.$$

These are the integer division and remainder function as defined by the C99 standard [ISO/IEC 9899:1999(E), Programming Languages - C (ISO and ANSI C99 Standard)].

Proposition A If $x, y \in F_b$ and $y \neq 0$, then $x \operatorname{rem} y \in F_b$.

The proof is given in the next three lemmas.

Lemma 1 Let $G_b = \{ n \in F_b^+ \mid n \text{ is odd} \}$. Then $G_b = \{1, 3, 5, \ldots, 2^{b+1} - 1\}$. Furthermore, if $x \in F_b^+$ then there exist $n \in G_b$ and $f \in \mathbb{N}$ such that $x = n \cdot 2^f$.

Proof Let $n \in \{1, 3, 5, \ldots, 2^{b+1} - 1\}$. There is a non negative integer $\beta \leq b$ such that $2^\beta \leq n < 2^{\beta+1}$. Then $n = (1 + m/2^b) \cdot 2^e$ with $m = (n - 2^\beta) \cdot 2^{b-\beta}$ and $e = \beta$. Here $m < (2^{\beta+1} - 2^\beta) \cdot 2^{b-\beta} = 2^b$ so that $n \in G_b$. The same argument shows that odd integers larger than 2^{b+1} do not in fact belong to G_b, since the corresponding value of m would exceed the bound $2^b - 1$ in the definition.

For the second part, let $x = (1 + m/2^b) \cdot 2^e \in F_b^+$. Let $m = 2^d \cdot m_1$ with m_1 odd and $d < b$. Then $n = 2^{b-d} + m_1$ is an odd integer that belongs to G_b since $2^{b-d} + m_1 \leq 2^{b-d} + (2^b - 1)/2^d < 2^{b-d+1} \leq 2^{b+1}$, using the first part. Hence we may take $f = e + d - b$ which is non negative since otherwise $m \cdot 2^{e-b} = m_1 \cdot 2^{e+d-b}$ would not be an integer as assumed.

Lemma 2 If $x, y \in F_b^+$ and y does not divide x, then $x \bmod y \in F_b^+$.

Proof By Lemma 1 above we may assume that $x = n \cdot 2^e$ and $y = m \cdot 2^f$ with $n, m \in G_b$ odd integers, and $e, f \in \mathbb{N}$. Let $k = \lfloor x/y \rfloor$. The goal is to prove that $x - ky \in F_b^+$: we may assume that $k > 0$, that is, that $x > y$ for otherwise $x \bmod y = x$ and there is nothing to prove.

- If $e < f$ then $x - ky = 2^e(n - km \cdot 2^{f-e})$. The integer $n - km \cdot 2^{f-e}$ is positive, odd and smaller than n, and therefore belongs to G_b.
- If $e = f$ then $x - ky = 2^e(n - km)$. The integer $n - km$ is positive and smaller than n, and therefore belongs to F_b^+.
- If $e > f$ then $x - ky = 2^f(n \cdot 2^{e-f} - km)$. The integer $n \cdot 2^{e-f} - km$ is positive and smaller than m, and therefore belongs to F_b^+: in fact

$$n \cdot 2^{e-f} - km = n \cdot 2^{e-f} - \left\lfloor \frac{x}{y} \right\rfloor m = n \cdot 2^{e-f} - \left\lfloor \frac{n \cdot 2^{e-f}}{m} \right\rfloor m.$$

  In other words, this integer is $n \cdot 2^{e-f} \bmod m$ and therefore it is smaller than m.

In all cases, we wrote $x - ky$ as the product of a power of 2 and an element of F_b^+, and this product is another element of F_b^+.

Lemma 3 For $x, y \in \mathbb{Z}$ with $y \neq 0$, we have

$$x \operatorname{rem} y = \begin{cases} x \operatorname{rem} |y|, & \text{if } x \geq 0; \\ -(|x| \operatorname{rem} |y|), & \text{if } x < 0. \end{cases}$$

Proof Throughout the proof we write $x_0 = |x|$ and $y_0 = |y|$. First, assume that $x \geq 0$ and that $y < 0$. Let $k = \phi(x/y_0) = -\phi(x/y)$, by the property above. We have

$$x \operatorname{rem} y = x - (x \operatorname{div} y)y = x - (-k)y = x - k(-y) = x - ky_0.$$

Next, assume that $x < 0$ and that $y < 0$. Let $k = \phi(x_0/y_0) = \phi(x/y)$. We have

$$x \operatorname{rem} y = x - (x \operatorname{div} y)y = x - ky = -(-x - k(-y)) = -(x_0 - ky_0).$$

Finally, assume that $x < 0$ and that $y > 0$. Let $k = \phi(x_0/y) = -\phi(x/y)$, again by the property above. We have

$$x \operatorname{rem} y = x - (x \operatorname{div} y)y = x - (-k)y = -(-x - ky) = -(x_0 - ky).$$

This completes the proof.

Lemma 4 If $x, y \in F_b^+$ then $\gcd(x, y) \in F_b^+$.

Proof Let $x = n \cdot 2^e$ and $y = m \cdot 2^f$ with $n, m \in G_b$ odd integers, and $e, f \in \mathbb{N}$. Then $\gcd(x, y) = \gcd(n, m) \cdot 2^{\min(e,f)}$, and therefore it belongs to F_b^+, since $\gcd(n, m) \leq \min(n, m)$ so that it belongs to G_b.

Lemma 5 If $x, y \in F_b^+$, then $x/\gcd(x, y) \in F_b^+$.

Proof With the same notation as in the previous Lemma, both n and $\gcd(n, m) \in G_b$: but all positive odd integers up to and including n belong to G_b, so that $n/\gcd(n, m)$ does as well. By Lemma 1 $x/\gcd(x, y) = n \cdot 2^e/(\gcd(n, m) \cdot 2^{\min(e,f)}) = (n/\gcd(n, m)) \cdot 2^{e-\min(e,f)} \in F_b^+$.

2 GNU General Public License

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other

parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material

or in the Appropriate Legal Notices displayed by works containing it; or

- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the

exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS

AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
 one line to give the program's name and a brief idea of what it does.
Copyright (C)  year   name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see  http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
 program  Copyright (C)  year   name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

3 GNU Free Documentation License

Version 1.2, November 2002

```
Copyright (C) 2000,2001,2002  Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA  02110-1301,  USA.
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is

addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but

different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled
"GNU Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

4 Module Index

4.1 Modules

Here is a list of all modules:

C++ Language Interface 89

5 Namespace Index

5.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Parma_Polyhedra_Library (The entire library is confined to this namespace) 104

Parma_Polyhedra_Library::Checked (Types and functions implementing checked numbers) 166

Parma_Polyhedra_Library::Implementation (Implementation related data and functions) 225

Parma_Polyhedra_Library::Implementation::Octagonal_Shapes 226

Parma_Polyhedra_Library::Implementation::Pointset_Powersets 226

Parma_Polyhedra_Library::Implementation::Termination 226

Parma_Polyhedra_Library::Interfaces (Data and functions related to language interfaces) 231

Parma_Polyhedra_Library::Interfaces::C (Data and functions related to the C language interface) 232

Parma_Polyhedra_Library::Interfaces::Java (Data and functions related to the Java language interface) 232

Parma_Polyhedra_Library::Interfaces::OCaml (Data and functions related to the OCaml language interface) 233

Parma_Polyhedra_Library::Interfaces::Prolog (Data and functions related to the Prolog language interfaces) 233

Parma_Polyhedra_Library::Interfaces::Prolog::Ciao (Data and functions related to the Ciao Prolog language interface) 233

Parma_Polyhedra_Library::Interfaces::Prolog::GNU (Data and functions related to the GNU Prolog language interface) 234

Parma_Polyhedra_Library::Interfaces::Prolog::SICStus (Data and functions related to the SICStus language interface) 234

Parma_Polyhedra_Library::Interfaces::Prolog::SWI (Data and functions related to the SWI-Prolog language interface) 234

Parma_Polyhedra_Library::Interfaces::Prolog::XSB (Data and functions related to the XSB language interface) 234

Parma_Polyhedra_Library::Interfaces::Prolog::YAP (Data and functions related to the YAP language interface) 234

Parma_Polyhedra_Library::Interval_NS 235

Parma_Polyhedra_Library::IO_Operators (All input/output operators are confined to this namespace) 240

std (The standard C++ namespace) 241

6 Class Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any 241

Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U > 242

Parma_Polyhedra_Library::BD_Shape< T > 258

Parma_Polyhedra_Library::BHRZ03_Certificate 337

Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign > 342

Parma_Polyhedra_Library::Bit_Matrix 344

Parma_Polyhedra_Library::Bit_Row 354

Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than 363

Parma_Polyhedra_Library::Bool< b > 364

Parma_Polyhedra_Library::Bool< false > 364

Parma_Polyhedra_Library::False 652

Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 > 1571

Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable > 1574

Parma_Polyhedra_Library::Fit< T, v, Enable > 653

Parma_Polyhedra_Library::Float< T > 668

Parma_Polyhedra_Library::Has_OK< T, Enable > 879

Parma_Polyhedra_Library::Is_Checked< T > 952

Parma_Polyhedra_Library::Is_Native< T > 956

Parma_Polyhedra_Library::Is_Same< T1, T2 > 978

Parma_Polyhedra_Library::Is_Special< T > 985

Parma_Polyhedra_Library::Bool< Is_Native< T >::value||Is_Checked< T >::value > 364

Parma_Polyhedra_Library::Is_Native_Or_Checked< T > 977

Parma_Polyhedra_Library::Is_Singleton< T, Enable > 983

Parma_Polyhedra_Library::Bool< Slow_Copy< T >::value > 364

Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > > 1597

Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > > 1598

Parma_Polyhedra_Library::Bool< true > 364

Parma_Polyhedra_Library::True 1663

Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T > 1573

Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::*)() const,&T::OK >::type > 881

Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > > 953

Parma_Polyhedra_Library::Is_Native< char > 957

Parma_Polyhedra_Library::Is_Native< mpq_class > 959

Parma_Polyhedra_Library::Is_Native< mpz_class > 960

Parma_Polyhedra_Library::Is_Native< signed char > 962

Parma_Polyhedra_Library::Is_Native< signed int > 963

Parma_Polyhedra_Library::Is_Native< signed long > 965

Parma_Polyhedra_Library::Is_Native< signed long long > 966

Parma_Polyhedra_Library::Is_Native< signed short > 968

Parma_Polyhedra_Library::Is_Native< unsigned char > 969

Parma_Polyhedra_Library::Is_Native< unsigned int > 971

Parma_Polyhedra_Library::Is_Native< unsigned long > 972

Parma_Polyhedra_Library::Is_Native< unsigned long long > 974

Parma_Polyhedra_Library::Is_Native< unsigned short > 975

Parma_Polyhedra_Library::Is_Same< T, T > 979

Parma_Polyhedra_Library::Is_Special< Minus_Infinity > 986

Parma_Polyhedra_Library::Is_Special< Not_A_Number > 988

Parma_Polyhedra_Library::Is_Special< Plus_Infinity > 989

Parma_Polyhedra_Library::Boundary_Value< T, Enable > 365

Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type > 366

Parma_Polyhedra_Library::Box< ITV > 366

Parma_Polyhedra_Library::Check_Overflow_Policy< T > 444

Parma_Polyhedra_Library::Checked_Number< T, Policy > 446

Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T > 475

Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient > 480

Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer > 480

Parma_Polyhedra_Library::BHRZ03_Certificate::Compare 481

Parma_Polyhedra_Library::H79_Certificate::Compare 482

Parma_Polyhedra_Library::Grid_Certificate::Compare 482

Parma_Polyhedra_Library::Variable::Compare 483

Parma_Polyhedra_Library::Compile_Time_Check< true > 484

Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 > 523

Parma_Polyhedra_Library::Matrix::const_iterator 525

Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator 529

Parma_Polyhedra_Library::Constraint_System::const_iterator 534

Parma_Polyhedra_Library::Congruence_System::const_iterator 539

Parma_Polyhedra_Library::Generator_System::const_iterator 544

Parma_Polyhedra_Library::Grid_Generator_System::const_iterator 549

Parma_Polyhedra_Library::const_iterator_to_const< Container > 554

Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 > 606

Parma_Polyhedra_Library::DB_Matrix< T > 608

Parma_Polyhedra_Library::DB_Row_Impl_Handler< T > 632

Parma_Polyhedra_Library::DB_Row< T > 619

Parma_Polyhedra_Library::Determinate< PSET > 635

Parma_Polyhedra_Library::Domain_Product< D1, D2 > 644

Parma_Polyhedra_Library::Enable_If< b, T > 646

Parma_Polyhedra_Library::Enable_If< true, T > 646

Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T > 647

Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp > 648

Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type > 648

Parma_Polyhedra_Library::Extended_Number_Policy 650

Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type > 655

Parma_Polyhedra_Library::Row::Flags 655

Parma_Polyhedra_Library::Linear_Row::Flags 661

Parma_Polyhedra_Library::Checked::Float_2exp 670

Parma_Polyhedra_Library::float_ibm_double 670

Parma_Polyhedra_Library::float_ibm_single 671

Parma_Polyhedra_Library::float_ieee754_double 676

Parma_Polyhedra_Library::float_ieee754_half 681

Parma_Polyhedra_Library::float_ieee754_quad 685

Parma_Polyhedra_Library::float_ieee754_single 690

Parma_Polyhedra_Library::float_intel_double_extended 695

Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > > 700

Parma_Polyhedra_Library::GMP_Integer 745

Parma_Polyhedra_Library::Grid 749

Parma_Polyhedra_Library::Grid_Certificate 832

Parma_Polyhedra_Library::H79_Certificate 876

Parma_Polyhedra_Library::Row_Impl_Handler::Impl 882

Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl 890

Parma_Polyhedra_Library::Init 898

Parma_Polyhedra_Library::Interval_Base 930

Parma_Polyhedra_Library::Interval< Boundary, Info > 900

Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > 930

Parma_Polyhedra_Library::Interval_Info_Null< Policy > 937

Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy > 941

Parma_Polyhedra_Library::Interval_Restriction_Integer_Base 945

Parma_Polyhedra_Library::Interval_Restriction_Integer< Base > 943

Parma_Polyhedra_Library::Simple_Restriction_Integer 1593

Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base 949

Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base > 946

Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T > 1595

Parma_Polyhedra_Library::Interval_Restriction_None_Base 951

Parma_Polyhedra_Library::Interval_Restriction_None< Base > 950

Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived > 981

Parma_Polyhedra_Library::Is_Same_Or_Derived< Interval_Base, T > 981

Parma_Polyhedra_Library::Is_Interval< T > 955

Parma_Polyhedra_Library::is_signed||sizeof(T1)< sizeof(T2)||sizeof(T1)< sizeof(int)))> 364

Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type > 1576

Parma_Polyhedra_Library::iterator_to_const< Container > 991

Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp > 997

Parma_Polyhedra_Library::Checked::Larger< char > 997

Parma_Polyhedra_Library::Checked::Larger< signed char > 999

Parma_Polyhedra_Library::Checked::Larger< signed int > 1001

Parma_Polyhedra_Library::Checked::Larger< signed long > 1002

Parma_Polyhedra_Library::Checked::Larger< signed long long > 1004

Parma_Polyhedra_Library::Checked::Larger< signed short > 1005

Parma_Polyhedra_Library::Checked::Larger< unsigned char > 1007

Parma_Polyhedra_Library::Checked::Larger< unsigned int > 1009

Parma_Polyhedra_Library::Checked::Larger< unsigned long > 1010

Parma_Polyhedra_Library::Checked::Larger< unsigned long long > 1012

Parma_Polyhedra_Library::Checked::Larger< unsigned short > 1013

Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS > 1015

Parma_Polyhedra_Library::Matrix 1078

Parma_Polyhedra_Library::Congruence_System 504

Parma_Polyhedra_Library::Linear_System 1056

Parma_Polyhedra_Library::Constraint_System 588

Parma_Polyhedra_Library::Generator_System 726

Parma_Polyhedra_Library::Grid_Generator_System 857

Parma_Polyhedra_Library::maybe_assign_struct< To, From > 1097

Parma_Polyhedra_Library::maybe_assign_struct< Type, Type > 1097

Parma_Polyhedra_Library::Minus_Infinity 1098

Parma_Polyhedra_Library::MIP_Problem 1098

Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > > 1131

Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > 1132

Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > > 1133

Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > 1134

Parma_Polyhedra_Library::PIP_Solution_Node::No_Constraints 1143

Parma_Polyhedra_Library::No_Reduction< D1, D2 > 1143

Parma_Polyhedra_Library::Not_A_Number 1144

Parma_Polyhedra_Library::Checked::number_struct 1145

Parma_Polyhedra_Library::Octagonal_Shape< T > 1146

Parma_Polyhedra_Library::OR_Matrix< T > 1227

Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > 1245

Parma_Polyhedra_Library::PIP_Problem 1297

Parma_Polyhedra_Library::PIP_Tree_Node 1333

Parma_Polyhedra_Library::PIP_Decision_Node 1289

Parma_Polyhedra_Library::PIP_Solution_Node 1321

Parma_Polyhedra_Library::Plus_Infinity 1347

Parma_Polyhedra_Library::Poly_Con_Relation 1399

Parma_Polyhedra_Library::Poly_Gen_Relation 1407

Parma_Polyhedra_Library::Polyhedron 1414

Parma_Polyhedra_Library::C_Polyhedron 436

Parma_Polyhedra_Library::NNC_Polyhedron 1135

Parma_Polyhedra_Library::Powerset< D > 1514

Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > > 1514

Parma_Polyhedra_Library::Pointset_Powerset< PSET > 1347

Parma_Polyhedra_Library::Interval_NS::Property 1534

Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U > 1536

Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation 1539

Parma_Polyhedra_Library::Rectilinear_Distance_Specialization< Temp > 1541

Parma_Polyhedra_Library::Recycle_Input 1541

Parma_Polyhedra_Library::Determinate< PSET >::Rep 1542

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > 1547

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > 1548

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > 1549

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > 1550

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > 1551

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > 1552

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > 1553

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > 1554

Parma_Polyhedra_Library::Row_Impl_Handler 1568

Parma_Polyhedra_Library::Row 1555

Parma_Polyhedra_Library::Congruence 484

Parma_Polyhedra_Library::Linear_Row 1039

Parma_Polyhedra_Library::Constraint 563

Parma_Polyhedra_Library::Generator 700

Parma_Polyhedra_Library::Grid_Generator 835

Parma_Polyhedra_Library::Linear_Expression 1017

Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter 251

Parma_Polyhedra_Library::Linear_System::Row_Less_Than 1571

Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy 1576

Parma_Polyhedra_Library::Scalar_Products 1577

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type > 1585

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char > 1586

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float > 1586

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char > 1587

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int > 1587

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long > 1588

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short > 1588

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char > 1589

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int > 1589

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long > 1590

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long > 1591

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short > 1591

Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 > 1592

Parma_Polyhedra_Library::Smash_Reduction< D1, D2 > 1599

Parma_Polyhedra_Library::Grid::Status 1600

Parma_Polyhedra_Library::Box< ITV >::Status 1610

Parma_Polyhedra_Library::BD_Shape< T >::Status 1615

Status 1622

Parma_Polyhedra_Library::Polyhedron::Status 1630

Parma_Polyhedra_Library::Octagonal_Shape< T >::Status 1644

Parma_Polyhedra_Library::stdiobuf 1649

Parma_Polyhedra_Library::PIP_Solution_Node::Tableau 1653

Parma_Polyhedra_Library::TConstant< T, v > 1658

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable > 561

Parma_Polyhedra_Library::TConstant< C_Integer< T >::smaller_signed_type, v > 1658

Parma_Polyhedra_Library::Constant_< C_Integer< T >::smaller_signed_type, v, prefer_signed > 561

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type > 562

Parma_Polyhedra_Library::TConstant< C_Integer< T >::smaller_unsigned_type, v > 1658

Parma_Polyhedra_Library::Constant_< C_Integer< T >::smaller_unsigned_type, v, prefer_signed > 561

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type > 562

Parma_Polyhedra_Library::TConstant< long long, v > 1658

Parma_Polyhedra_Library::Constant_< long long, v, prefer_signed > 561

Parma_Polyhedra_Library::Constant< v, prefer_signed > 560

Parma_Polyhedra_Library::Throwable 1659

Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign 1661

Parma_Polyhedra_Library::Variable 1664

Parma_Polyhedra_Library::Variables_Set 1671

Parma_Polyhedra_Library::Weightwatch_Traits 1676

Parma_Polyhedra_Library::Widening_Function< PSET > 1678

Parma_Polyhedra_Library::Linear_System::With_Pending 1680

7 Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any (A class that is constructible from just anything) 241

Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U > (A template class to derive both OR_Matrix::iterator and OR_Matrix::const_iterator) 242

Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter (Artificial parameters in PIP solution trees) 251

Parma_Polyhedra_Library::BD_Shape< T > (A bounded difference shape) 258

Parma_Polyhedra_Library::BHRZ03_Certificate (The convergence certificate for the BHRZ03 widening operator) 337

Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign > (A function adapter for the Determinate class) 342

Parma_Polyhedra_Library::Bit_Matrix (A matrix of bits) 344

Parma_Polyhedra_Library::Bit_Row (A row in a matrix of bits) 354

Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than (Ordering predicate (used when implementing the sort algorithm)) 363

Parma_Polyhedra_Library::Bool< b > (A class holding a constant called `value` that evaluates to `b`) 364

Parma_Polyhedra_Library::Boundary_Value< T, Enable > 365

Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type > 366

Parma_Polyhedra_Library::Box< ITV > (A not necessarily closed, iso-oriented hyperrectangle) 366

Parma_Polyhedra_Library::C_Polyhedron (A closed convex polyhedron) 436

Parma_Polyhedra_Library::Check_Overflow_Policy< T > (A policy checking for overflows) 444

Parma_Polyhedra_Library::Checked_Number< T, Policy > (A wrapper for numeric types implementing a given policy) 446

Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T > 475

Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient > (Coefficient traits) 480

Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer > (Coefficient traits specialization for unbounded integers) 480

Parma_Polyhedra_Library::BHRZ03_Certificate::Compare (A total ordering on BHRZ03 certificates) 481

Parma_Polyhedra_Library::H79_Certificate::Compare (A total ordering on H79 certificates) 482

Parma_Polyhedra_Library::Grid_Certificate::Compare (A total ordering on Grid certificates) 482

Parma_Polyhedra_Library::Variable::Compare (Binary predicate defining the total ordering on variables) 483

Parma_Polyhedra_Library::Compile_Time_Check< true > (A class that is only defined if `b` evaluates to `true`) 484

Parma_Polyhedra_Library::Congruence (A linear congruence) 484

Parma_Polyhedra_Library::Congruence_System (A system of congruences) 504

Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 > (This class provides the reduction method for the Congruences_Product domain) 523

Parma_Polyhedra_Library::Matrix::const_iterator (An iterator over a matrix) 525

Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator (A read-only iterator over the rows of the matrix) 529

Parma_Polyhedra_Library::Constraint_System::const_iterator (An iterator over a system of constraints) 534

Parma_Polyhedra_Library::Congruence_System::const_iterator (An iterator over a system of congruences) 539

Parma_Polyhedra_Library::Generator_System::const_iterator (An iterator over a system of generators) 544

Parma_Polyhedra_Library::Grid_Generator_System::const_iterator (An iterator over a system of grid generators) 549

Parma_Polyhedra_Library::const_iterator_to_const< Container > (A const_iterator on a sequence of read-only objects) 554

Parma_Polyhedra_Library::Constant< v, prefer_signed > 560

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable > 561

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type > 562

Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type > 562

Parma_Polyhedra_Library::Constraint (A linear equality or inequality) 563

Parma_Polyhedra_Library::Constraint_System (A system of constraints) 588

Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 > (This class provides the reduction method for the Constraints_Product domain) 606

Parma_Polyhedra_Library::DB_Matrix< T > (The base class for the square matrices) 608

Parma_Polyhedra_Library::DB_Row< T > (The base class for the single rows of matrices) 619

Parma_Polyhedra_Library::DB_Row_Impl_Handler< T > (The handler of the actual DB_Row implementation) 632

Parma_Polyhedra_Library::Determinate< PSET > (A wrapper for PPL pointsets, providing them with a *determinate constraint system* interface, as defined in [Bag98]) 635

Parma_Polyhedra_Library::Domain_Product< D1, D2 > (This class is temporary and will be removed when template typedefs will be supported in C++) 644

Parma_Polyhedra_Library::Enable_If< b, T > (A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`) 646

Parma_Polyhedra_Library::Enable_If< true, T > (A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`) 646

Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T > 647

Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp > 648

Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type > 648

Parma_Polyhedra_Library::Extended_Number_Policy 650

Parma_Polyhedra_Library::False (A class holding a constant called `value` that evaluates to `false`) 652

Parma_Polyhedra_Library::Fit< T, v, Enable > 653

Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type > 655

Parma_Polyhedra_Library::Row::Flags (Wrapper class to represent a set of flags with bits in a native unsigned integral type) 655

Parma_Polyhedra_Library::Linear_Row::Flags (The type of the object to which the coefficients refer to, encoding both topology and kind) 661

Parma_Polyhedra_Library::Float< T > 668

Parma_Polyhedra_Library::Checked::Float_2exp 670

Parma_Polyhedra_Library::float_ibm_double 670

Parma_Polyhedra_Library::float_ibm_single 671

Parma_Polyhedra_Library::float_ieee754_double 676

Parma_Polyhedra_Library::float_ieee754_half 681

Parma_Polyhedra_Library::float_ieee754_quad 685

Parma_Polyhedra_Library::float_ieee754_single 690

Parma_Polyhedra_Library::float_intel_double_extended 695

Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > > 700

Parma_Polyhedra_Library::Generator (A line, ray, point or closure point) 700

Parma_Polyhedra_Library::Generator_System (A system of generators) 726

Parma_Polyhedra_Library::GMP_Integer (Unbounded integers as provided by the GMP library) 745

Parma_Polyhedra_Library::Grid (A grid) 749

Parma_Polyhedra_Library::Grid_Certificate (The convergence certificate for the Grid widening operator) 832

Parma_Polyhedra_Library::Grid_Generator (A grid line, parameter or grid point) 835

Parma_Polyhedra_Library::Grid_Generator_System (A system of grid generators) 857

Parma_Polyhedra_Library::H79_Certificate (A convergence certificate for the H79 widening operator) 876

Parma_Polyhedra_Library::Has_OK< T, Enable > 879

Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::*)() const,&T::OK >::type > 881

Parma_Polyhedra_Library::Row_Impl_Handler::Impl (The actual implementation of a Row object) 882

Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl (The real implementation of a DB_Row object) 890

Parma_Polyhedra_Library::Init (Class for initialization and finalization) 898

Parma_Polyhedra_Library::Interval< Boundary, Info > (A generic, not necessarily closed, possibly restricted interval) 900

Parma_Polyhedra_Library::Interval_Base 930

Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > 930

Parma_Polyhedra_Library::Interval_Info_Null< Policy > 937

Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy > 941

Parma_Polyhedra_Library::Interval_Restriction_Integer< Base > 943

Parma_Polyhedra_Library::Interval_Restriction_Integer_Base 945

Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base > 946

Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base 949

Parma_Polyhedra_Library::Interval_Restriction_None< Base > 950

Parma_Polyhedra_Library::Interval_Restriction_None_Base 951

Parma_Polyhedra_Library::Is_Checked< T > 952

Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > > 953

Parma_Polyhedra_Library::Is_Interval< T > 955

Parma_Polyhedra_Library::Is_Native< T > 956

Parma_Polyhedra_Library::Is_Native< char > 957

Parma_Polyhedra_Library::Is_Native< mpq_class > 959

Parma_Polyhedra_Library::Is_Native< mpz_class > 960

Parma_Polyhedra_Library::Is_Native< signed char > 962

Parma_Polyhedra_Library::Is_Native< signed int > 963

Parma_Polyhedra_Library::Is_Native< signed long > 965

Parma_Polyhedra_Library::Is_Native< signed long long > 966

Parma_Polyhedra_Library::Is_Native< signed short > 968

Parma_Polyhedra_Library::Is_Native< unsigned char > 969

Parma_Polyhedra_Library::Is_Native< unsigned int > 971

Parma_Polyhedra_Library::Is_Native< unsigned long > 972

Parma_Polyhedra_Library::Is_Native< unsigned long long > 974

Parma_Polyhedra_Library::Is_Native< unsigned short > 975

Parma_Polyhedra_Library::Is_Native_Or_Checked< T > 977

Parma_Polyhedra_Library::Is_Same< T1, T2 > (A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`) 978

Parma_Polyhedra_Library::Is_Same< T, T > (A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`) 979

Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived > (A class holding a constant called `value` that evaluates to `true` if and only if `Base` is the same type as `Derived` or `Derived` is a class derived from `Base`) 981

Parma_Polyhedra_Library::Is_Singleton< T, Enable > 983

Parma_Polyhedra_Library::Is_Special< T > 985

Parma_Polyhedra_Library::Is_Special< Minus_Infinity > 986

Parma_Polyhedra_Library::Is_Special< Not_A_Number > 988

Parma_Polyhedra_Library::Is_Special< Plus_Infinity > 989

Parma_Polyhedra_Library::iterator_to_const< Container > (An iterator on a sequence of read-only objects) 991

Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp > 997

Parma_Polyhedra_Library::Checked::Larger< char > 997

Parma_Polyhedra_Library::Checked::Larger< signed char > 999

Parma_Polyhedra_Library::Checked::Larger< signed int > 1001

Parma_Polyhedra_Library::Checked::Larger< signed long > 1002

Parma_Polyhedra_Library::Checked::Larger< signed long long > 1004

Parma_Polyhedra_Library::Checked::Larger< signed short > 1005

Parma_Polyhedra_Library::Checked::Larger< unsigned char > 1007

Parma_Polyhedra_Library::Checked::Larger< unsigned int > 1009

Parma_Polyhedra_Library::Checked::Larger< unsigned long > 1010

Parma_Polyhedra_Library::Checked::Larger< unsigned long long > 1012

Parma_Polyhedra_Library::Checked::Larger< unsigned short > 1013

Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS > (Wraps a limited widening method into a function object) 1015

Parma_Polyhedra_Library::Linear_Expression (A linear expression) 1017

Parma_Polyhedra_Library::Linear_Row (The base class for linear expressions, constraints and generators) 1039

Parma_Polyhedra_Library::Linear_System (The base class for systems of constraints and generators) 1056

Parma_Polyhedra_Library::Matrix (A 2-dimensional matrix of coefficients) 1078

Parma_Polyhedra_Library::maybe_assign_struct< To, From > 1097

Parma_Polyhedra_Library::maybe_assign_struct< Type, Type > 1097

Parma_Polyhedra_Library::Minus_Infinity 1098

Parma_Polyhedra_Library::MIP_Problem (A Mixed Integer (linear) Programming problem) 1098

Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > > 1131

Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > 1132

Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > > 1133

Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > 1134

Parma_Polyhedra_Library::NNC_Polyhedron (A not necessarily closed convex polyhedron) 1135

Parma_Polyhedra_Library::PIP_Solution_Node::No_Constraints (A tag type to select the alternative copy constructor) 1143

Parma_Polyhedra_Library::No_Reduction< D1, D2 > (This class provides the reduction method for the Direct_Product domain) 1143

Parma_Polyhedra_Library::Not_A_Number 1144

Parma_Polyhedra_Library::Checked::number_struct 1145

Parma_Polyhedra_Library::Octagonal_Shape< T > (An octagonal shape) 1146

Parma_Polyhedra_Library::OR_Matrix< T > (A matrix representing octagonal constraints) 1227

Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > (The partially reduced product of two abstractions) 1245

Parma_Polyhedra_Library::PIP_Decision_Node (A tree node representing a decision in the space of solutions) 1289

Parma_Polyhedra_Library::PIP_Problem (A Parametric Integer (linear) Programming problem) 1297

Parma_Polyhedra_Library::PIP_Solution_Node (A tree node representing part of the space of solutions) 1321

Parma_Polyhedra_Library::PIP_Tree_Node (A node of the PIP solution tree) 1333

Parma_Polyhedra_Library::Plus_Infinity 1347

Parma_Polyhedra_Library::Pointset_Powerset< PSET > (The powerset construction instantiated on PPL pointset domains) 1347

Parma_Polyhedra_Library::Poly_Con_Relation (The relation between a polyhedron and a constraint) 1399

Parma_Polyhedra_Library::Poly_Gen_Relation (The relation between a polyhedron and a generator) 1407

Parma_Polyhedra_Library::Polyhedron (The base class for convex polyhedra) 1414

Parma_Polyhedra_Library::Powerset< D > (The powerset construction on a base-level domain) 1514

Parma_Polyhedra_Library::Interval_NS::Property 1534

Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U > (An object that behaves like a matrix's row with respect to the subscript operators) 1536

Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation (A helper class to temporarily relax a MIP problem using RAII) 1539

Parma_Polyhedra_Library::Rectilinear_Distance_Specialization< Temp > 1541

Parma_Polyhedra_Library::Recycle_Input (A tag class) 1541

Parma_Polyhedra_Library::Determinate< PSET >::Rep (The possibly shared representation of a Determinate object) 1542

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > 1547

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > 1548

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > 1549

Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > 1550

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > 1551

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > 1552

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > 1553

Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > 1554

Parma_Polyhedra_Library::Row (A finite sequence of coefficients) 1555

Parma_Polyhedra_Library::Row_Impl_Handler (The handler of the actual Row implementation) 1568

Parma_Polyhedra_Library::Linear_System::Row_Less_Than (Ordering predicate (used when implementing the sort algorithm)) 1571

Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 > 1571

Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T > 1573

Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable > 1574

Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type > 1576

Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy 1576

Parma_Polyhedra_Library::Scalar_Products (A class implementing various scalar product functions) 1577

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type > 1585

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char > 1586

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float > 1586

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char > 1587

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int > 1587

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long > 1588

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short > 1588

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char > 1589

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int > 1589

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long > 1590

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long > 1591

Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short > 1591

Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 > (This class provides the reduction method for the Shape_Preserving_Product domain) 1592

Parma_Polyhedra_Library::Simple_Restriction_Integer 1593

Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T > 1595

Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > > 1597

Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > > 1598

Parma_Polyhedra_Library::Smash_Reduction< D1, D2 > (This class provides the reduction method for the Smash_Product domain) 1599

Parma_Polyhedra_Library::Grid::Status (A conjunctive assertion about a grid) 1600

Parma_Polyhedra_Library::Box< ITV >::Status 1610

Parma_Polyhedra_Library::BD_Shape< T >::Status (A conjunctive assertion about a BD_Shape<T> object) 1615

Status (A conjunctive assertion about a polyhedron) 1622

Parma_Polyhedra_Library::Polyhedron::Status (A conjunctive assertion about a polyhedron) 1630

Parma_Polyhedra_Library::Octagonal_Shape< T >::Status (A conjunctive assertion about a Octagonal_Shape<T> object) 1644

Parma_Polyhedra_Library::stdiobuf 1649

Parma_Polyhedra_Library::PIP_Solution_Node::Tableau (The type for parametric simplex tableau) 1653

Parma_Polyhedra_Library::TConstant< T, v > 1658

Parma_Polyhedra_Library::Throwable (User objects the PPL can throw) 1659

Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign (Scalar product sign function object depending on topology) 1661

Parma_Polyhedra_Library::True (A class holding a constant called `value` that evaluates to `true`) 1663

Parma_Polyhedra_Library::Variable (A dimension of the vector space) 1664

Parma_Polyhedra_Library::Variables_Set (An std::set of variables' indexes) 1671

Parma_Polyhedra_Library::Weightwatch_Traits (Traits class for the deterministic timeout mechanism) 1676

Parma_Polyhedra_Library::Widening_Function< PSET > (Wraps a widening method into a function object) 1678

Parma_Polyhedra_Library::Linear_System::With_Pending (A tag class) 1680

8 File Index

8.1 File List

Here is a list of all files with brief descriptions:

algorithms.hh 1680

BD_Shape.cc 1681

BD_Shape.defs.hh 1681

BD_Shape.inlines.hh 1683

BD_Shape.templates.hh 1684

BD_Shape.types.hh 1686

BHRZ03_Certificate.cc 1686

BHRZ03_Certificate.defs.hh 1686

BHRZ03_Certificate.inlines.hh 1687

BHRZ03_Certificate.types.hh 1688

Bit_Matrix.cc 1688

Bit_Matrix.defs.hh 1689

Bit_Matrix.inlines.hh 1690

Bit_Matrix.types.hh 1691

Bit_Row.cc 1691

Bit_Row.defs.hh 1691

Bit_Row.inlines.hh 1692

Bit_Row.types.hh 1694

Box.cc 1694

Box.defs.hh 1694

Box.inlines.hh 1696

Box.templates.hh 1697

Box.types.hh 1699

C_Polyhedron.cc 1699

C_Polyhedron.defs.hh 1701

C_Polyhedron.inlines.hh 1702

C_Polyhedron.types.hh 1703

checked.cc 1703

checked.defs.hh 1704

checked.inlines.hh 1719

checked_float.inlines.hh 1723

checked_int.inlines.hh 1727

checked_mpq.inlines.hh 1733

checked_mpz.inlines.hh 1737

Checked_Number.cc 1740

Checked_Number.defs.hh 1740

Checked_Number.inlines.hh 1744

Checked_Number.types.hh 1748

Coefficient.cc 1749

Coefficient.defs.hh 1750

Coefficient.inlines.hh 1751

Coefficient.types.hh 1751

Coefficient_traits_template.hh 1752

Congruence.cc 1752

Congruence.defs.hh 1753

Congruence.inlines.hh 1754

Congruence.types.hh 1755

Congruence_System.cc 1755

Congruence_System.defs.hh 1756

Congruence_System.inlines.hh 1757

Congruence_System.types.hh 1758

Constraint.cc 1758

Constraint.defs.hh 1759

Constraint.inlines.hh 1760

Constraint.types.hh 1761

Constraint_System.cc 1761

Constraint_System.defs.hh 1761

Constraint_System.inlines.hh 1763

Constraint_System.types.hh 1763

conversion.cc 1764

DB_Matrix.defs.hh 1764

DB_Matrix.inlines.hh 1766

DB_Matrix.templates.hh 1767

DB_Matrix.types.hh 1768

DB_Row.defs.hh 1768

DB_Row.inlines.hh 1770

DB_Row.templates.hh 1771

DB_Row.types.hh 1772

Determinate.defs.hh 1772

Determinate.inlines.hh 1774

distances.defs.hh 1774

distances.inlines.hh 1775

distances.types.hh 1776

Float.cc 1777

Float.defs.hh 1777

Float.inlines.hh 1779

Generator.cc 1779

Generator.defs.hh 1780

Generator.inlines.hh 1781

Generator.types.hh 1782

Generator_System.cc 1782

Generator_System.defs.hh 1782

Generator_System.inlines.hh 1784

Generator_System.types.hh 1784

globals.cc 1785

globals.defs.hh 1785

globals.inlines.hh 1792

globals.types.hh 1793

GMP_Integer.defs.hh 1795

GMP_Integer.inlines.hh 1796

GMP_Integer.types.hh 1796

Grid.defs.hh 1797

Grid.inlines.hh 1799

Grid.templates.hh 1800

Grid.types.hh 1800

Grid_Certificate.cc 1801

Grid_Certificate.defs.hh 1801

Grid_Certificate.inlines.hh 1802

Grid_Certificate.types.hh 1803

Grid_chdims.cc 1803

Grid_conversion.cc 1803

Grid_Generator.cc 1804

Grid_Generator.defs.hh 1804

Grid_Generator.inlines.hh 1805

Grid_Generator.types.hh 1806

Grid_Generator_System.cc 1806

Grid_Generator_System.defs.hh 1806

Grid_Generator_System.inlines.hh 1807

Grid_Generator_System.types.hh 1808

Grid_nonpublic.cc 1809

Grid_public.cc 1809

Grid_simplify.cc 1810

Grid_widenings.cc 1810

H79_Certificate.cc 1811

H79_Certificate.defs.hh 1811

H79_Certificate.inlines.hh 1812

H79_Certificate.types.hh 1813

Init.cc 1813

Init.defs.hh 1815

Init.inlines.hh 1816

Init.types.hh 1817

initializer.hh 1817

Interval.defs.hh 1818

Interval.inlines.hh 1820

Interval.types.hh 1823

Interval_Info.defs.hh 1823

Interval_Info.inlines.hh 1825

Interval_Info.types.hh 1825

Interval_Restriction.defs.hh 1826

iterator_to_const.defs.hh 1829

iterator_to_const.inlines.hh 1831

iterator_to_const.types.hh 1831

Linear_Expression.cc 1832

Linear_Expression.defs.hh 1832

Linear_Expression.inlines.hh 1833

Linear_Expression.types.hh 1834

Linear_Row.cc 1835

Linear_Row.defs.hh 1835

Linear_Row.inlines.hh 1836

Linear_Row.types.hh 1837

Linear_System.cc 1837

Linear_System.defs.hh 1838

Linear_System.inlines.hh 1839

Linear_System.types.hh 1840

math_utilities.defs.hh 1841

math_utilities.inlines.hh 1842

Matrix.cc 1844

Matrix.defs.hh 1844

Matrix.inlines.hh 1845

Matrix.types.hh 1846

meta_programming.hh 1846

minimize.cc 1849

MIP_Problem.cc 1849

MIP_Problem.defs.hh 1850

MIP_Problem.inlines.hh 1851

MIP_Problem.templates.hh 1852

MIP_Problem.types.hh 1854

namespaces.hh 1854

NNC_Polyhedron.cc 1855

NNC_Polyhedron.defs.hh 1856

NNC_Polyhedron.inlines.hh 1857

NNC_Polyhedron.types.hh 1858

Octagonal_Shape.cc 1858

Octagonal_Shape.defs.hh 1858

Octagonal_Shape.inlines.hh 1861

Octagonal_Shape.templates.hh 1862

Octagonal_Shape.types.hh 1863

OR_Matrix.defs.hh 1863

OR_Matrix.inlines.hh 1865

OR_Matrix.templates.hh 1866

OR_Matrix.types.hh 1868

Partially_Reduced_Product.defs.hh 1868

Partially_Reduced_Product.inlines.hh 1870

Partially_Reduced_Product.templates.hh 1871

Partially_Reduced_Product.types.hh 1872

Ph_Status.cc 1873

Ph_Status.idefs.hh 1873

Ph_Status.inlines.hh 1873

PIP_Problem.cc 1873

PIP_Problem.defs.hh 1874

PIP_Problem.inlines.hh 1876

PIP_Problem.templates.hh 1876

PIP_Problem.types.hh 1877

PIP_Tree.cc 1877

PIP_Tree.defs.hh 1878

PIP_Tree.inlines.hh 1880

PIP_Tree.types.hh 1880

Pointset_Powerset.cc 1880

Pointset_Powerset.defs.hh 1881

Pointset_Powerset.inlines.hh 1882

Pointset_Powerset.templates.hh 1883

Pointset_Powerset.types.hh 1885

Poly_Con_Relation.cc 1885

Poly_Con_Relation.defs.hh 1885

Poly_Con_Relation.inlines.hh 1887

Poly_Con_Relation.types.hh 1887

Poly_Gen_Relation.cc 1887

Poly_Gen_Relation.defs.hh 1888

Poly_Gen_Relation.inlines.hh 1889

Poly_Gen_Relation.types.hh 1889

Polyhedron.defs.hh 1890

Polyhedron.inlines.hh 1892

Polyhedron.templates.hh 1893

Polyhedron.types.hh 1894

Polyhedron_chdims.cc 1894

Polyhedron_nonpublic.cc 1894

Polyhedron_public.cc 1895

Polyhedron_widenings.cc 1896

Powerset.defs.hh 1898

Powerset.inlines.hh 1900

Powerset.templates.hh 1900

Powerset.types.hh 1902

Row.cc 1902

Row.defs.hh 1903

Row.inlines.hh 1904

Row.types.hh 1905

Scalar_Products.cc 1905

Scalar_Products.defs.hh 1905

Scalar_Products.inlines.hh 1906

Scalar_Products.types.hh 1907

simplify.cc 1907

stdiobuf.cc 1908

stdiobuf.defs.hh 1908

stdiobuf.inlines.hh 1909

stdiobuf.types.hh 1909

termination.cc 1910

termination.defs.hh 1911

termination.templates.hh 1914

Topology.hh 1917

Variable.cc **1918**

Variable.defs.hh **1918**

Variable.inlines.hh **1919**

Variable.types.hh **1920**

Variables_Set.cc **1920**

Variables_Set.defs.hh **1921**

Variables_Set.inlines.hh **1922**

Variables_Set.types.hh **1923**

version.cc **1923**

version.hh **1924**

Widening_Function.defs.hh **1925**

Widening_Function.inlines.hh **1925**

Widening_Function.types.hh **1926**

9 Module Documentation

9.1 C++ Language Interface

The core implementation of the Parma Polyhedra Library is written in C++.

Classes

- struct Parma_Polyhedra_Library::Compile_Time_Check< true >
  A class that is only defined if `b` evaluates to `true`.
- struct Parma_Polyhedra_Library::Bool< b >
  A class holding a constant called `value` that evaluates to `b`.
- struct Parma_Polyhedra_Library::True
  A class holding a constant called `value` that evaluates to `true`.
- struct Parma_Polyhedra_Library::False
  A class holding a constant called `value` that evaluates to `false`.
- struct Parma_Polyhedra_Library::Is_Same< T1, T2 >
  A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`.
- struct Parma_Polyhedra_Library::Is_Same< T, T >
  A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`.

- struct Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `Base` *is the same type as* `Derived` *or* `Derived` *is a class derived from* `Base`*.*

- struct Parma_Polyhedra_Library::Enable_If< b, T >

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`*.*

- struct Parma_Polyhedra_Library::Enable_If< true, T >

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`*.*

- struct Parma_Polyhedra_Library::float_ieee754_half
- struct Parma_Polyhedra_Library::float_ieee754_single
- struct Parma_Polyhedra_Library::float_ieee754_double
- struct Parma_Polyhedra_Library::float_ibm_single
- struct Parma_Polyhedra_Library::float_ibm_double
- struct Parma_Polyhedra_Library::float_intel_double_extended
- struct Parma_Polyhedra_Library::float_ieee754_quad
- class Parma_Polyhedra_Library::Float< T >
- struct Parma_Polyhedra_Library::Extended_Number_Policy
- struct Parma_Polyhedra_Library::Check_Overflow_Policy< T >

  A policy checking for overflows.

- struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Is_Checked< T >
- struct Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Is_Native_Or_Checked< T >
- class Parma_Polyhedra_Library::Checked_Number< T, Policy >

  A wrapper for numeric types implementing a given policy.

- struct Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer >

  Coefficient traits specialization for unbounded integers.

- struct Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient >

  Coefficient traits.

- struct Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >
- class Parma_Polyhedra_Library::Init

  Class for initialization and finalization.

- struct Parma_Polyhedra_Library::Weightwatch_Traits

  Traits class for the deterministic timeout mechanism.

- class Parma_Polyhedra_Library::Throwable

  User objects the PPL can throw.

- struct Parma_Polyhedra_Library::Recycle_Input
  A tag class.
- class Parma_Polyhedra_Library::Row_Impl_Handler
  The handler of the actual Row implementation.
- class Parma_Polyhedra_Library::Row
  A finite sequence of coefficients.
- class Parma_Polyhedra_Library::Row::Flags
  Wrapper class to represent a set of flags with bits in a native unsigned integral type.
- class Parma_Polyhedra_Library::Row_Impl_Handler::Impl
  The actual implementation of a Row object.
- class Parma_Polyhedra_Library::Matrix
  A 2-dimensional matrix of coefficients.
- class Parma_Polyhedra_Library::Matrix::const_iterator
  An iterator over a matrix.
- class Parma_Polyhedra_Library::Linear_Row
  The base class for linear expressions, constraints and generators.
- class Parma_Polyhedra_Library::Linear_Row::Flags
  The type of the object to which the coefficients refer to, encoding both topology and kind.
- class Parma_Polyhedra_Library::Linear_System
  The base class for systems of constraints and generators.
- struct Parma_Polyhedra_Library::Linear_System::With_Pending
  A tag class.
- class Parma_Polyhedra_Library::MIP_Problem
  A Mixed Integer (linear) Programming problem.
- class Parma_Polyhedra_Library::Bit_Row
  A row in a matrix of bits.
- class Parma_Polyhedra_Library::Bit_Matrix
  A matrix of bits.
- struct Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than
  Ordering predicate (used when implementing the sort algorithm).
- class Parma_Polyhedra_Library::Variable
  A dimension of the vector space.
- struct Parma_Polyhedra_Library::Variable::Compare

Binary predicate defining the total ordering on variables.

- class Parma_Polyhedra_Library::Linear_Expression
  A linear expression.
- class Parma_Polyhedra_Library::Constraint
  A linear equality or inequality.
- class Parma_Polyhedra_Library::Congruence
  A linear congruence.
- class Parma_Polyhedra_Library::Generator
  A line, ray, point or closure point.
- class Parma_Polyhedra_Library::Grid_Generator
  A grid line, parameter or grid point.
- class Parma_Polyhedra_Library::Constraint_System
  A system of constraints.
- class Parma_Polyhedra_Library::Constraint_System::const_iterator
  An iterator over a system of constraints.
- class Parma_Polyhedra_Library::Congruence_System
  A system of congruences.
- class Parma_Polyhedra_Library::Congruence_System::const_iterator
  An iterator over a system of congruences.
- class Parma_Polyhedra_Library::Generator_System
  A system of generators.
- class Parma_Polyhedra_Library::Generator_System::const_iterator
  An iterator over a system of generators.
- class Parma_Polyhedra_Library::Grid_Generator_System
  A system of grid generators.
- class Parma_Polyhedra_Library::Grid_Generator_System::const_iterator
  An iterator over a system of grid generators.
- class Parma_Polyhedra_Library::Scalar_Products
  A class implementing various scalar product functions.
- class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign
  Scalar product sign function object depending on topology.
- class Status
  A conjunctive assertion about a polyhedron.

- class Parma_Polyhedra_Library::Polyhedron

  The base class for convex polyhedra.

- class Parma_Polyhedra_Library::Polyhedron::Status

  A conjunctive assertion about a polyhedron.

- class Parma_Polyhedra_Library::Grid

  A grid.

- class Parma_Polyhedra_Library::Poly_Con_Relation

  The relation between a polyhedron and a constraint.

- class Parma_Polyhedra_Library::Poly_Gen_Relation

  The relation between a polyhedron and a generator.

- class Parma_Polyhedra_Library::Interval< Boundary, Info >

  A generic, not necessarily closed, possibly restricted interval.

- class Parma_Polyhedra_Library::C_Polyhedron

  A closed convex polyhedron.

- class Parma_Polyhedra_Library::NNC_Polyhedron

  A not necessarily closed convex polyhedron.

- class Parma_Polyhedra_Library::Widening_Function< PSET >

  Wraps a widening method into a function object.

- class Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >

  Wraps a limited widening method into a function object.

- class Parma_Polyhedra_Library::BHRZ03_Certificate

  The convergence certificate for the BHRZ03 widening operator.

- struct Parma_Polyhedra_Library::BHRZ03_Certificate::Compare

  A total ordering on BHRZ03 certificates.

- class Parma_Polyhedra_Library::H79_Certificate

  A convergence certificate for the H79 widening operator.

- struct Parma_Polyhedra_Library::H79_Certificate::Compare

  A total ordering on H79 certificates.

- class Parma_Polyhedra_Library::Grid_Certificate

  The convergence certificate for the Grid widening operator.

- class Parma_Polyhedra_Library::Determinate< PSET >

  A wrapper for PPL pointsets, providing them with a determinate constraint system interface, as defined in [Bag98].

- class Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >

  A function adapter for the Determinate class.

- class Parma_Polyhedra_Library::Determinate< PSET >::Rep

  The possibly shared representation of a Determinate object.

- class Parma_Polyhedra_Library::iterator_to_const< Container >

  An iterator on a sequence of read-only objects.

- class Parma_Polyhedra_Library::const_iterator_to_const< Container >

  A const_iterator on a sequence of read-only objects.

- class Parma_Polyhedra_Library::Powerset< D >

  The powerset construction on a base-level domain.

- class Parma_Polyhedra_Library::Pointset_Powerset< PSET >

  The powerset construction instantiated on PPL pointset domains.

- class Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >

  This class provides the reduction method for the Smash_Product domain.

- class Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >

  This class provides the reduction method for the Constraints_Product domain.

- class Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >

  This class provides the reduction method for the Congruences_Product domain.

- class Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >

  This class provides the reduction method for the Shape_Preserving_Product domain.

- class Parma_Polyhedra_Library::No_Reduction< D1, D2 >

  This class provides the reduction method for the Direct_Product domain.

- class Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >

  The partially reduced product of two abstractions.

- class Parma_Polyhedra_Library::Box< ITV >

  A not necessarily closed, iso-oriented hyperrectangle.

- class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >

  The handler of the actual DB_Row implementation.

- class Parma_Polyhedra_Library::DB_Row< T >

  The base class for the single rows of matrices.

- class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl

  The real implementation of a DB_Row object.

- class Parma_Polyhedra_Library::DB_Matrix< T >

The base class for the square matrices.

- class Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator

  A read-only iterator over the rows of the matrix.

- class Parma_Polyhedra_Library::BD_Shape< T >

  A bounded difference shape.

- class Parma_Polyhedra_Library::BD_Shape< T >::Status

  A conjunctive assertion about a BD_Shape<T> object.

- class Parma_Polyhedra_Library::Octagonal_Shape< T >

  An octagonal shape.

- class Parma_Polyhedra_Library::PIP_Problem

  A Parametric Integer (linear) Programming problem.

- class Parma_Polyhedra_Library::GMP_Integer

  Unbounded integers as provided by the GMP library.

Namespaces

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_VERSION_MAJOR 0

  The major number of the PPL version.

- #define PPL_VERSION_MINOR 11

  The minor number of the PPL version.

- #define PPL_VERSION_REVISION 2

  The revision number of the PPL version.

- #define PPL_VERSION_BETA 0

  The beta number of the PPL version. This is zero for official releases and nonzero for development snapshots.

- #define PPL_VERSION "0.11.2"

  A string containing the PPL version.

- #define const_bool_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

  Declares a per-class constant of type `bool`, called `name` and with value `value`.

- #define const_int_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

  Declares a per-class constant of type `int`, called `name` and with value `value`.

- #define const_value_nodef(type, name, value)

  Declares a per-class constant of type `type`, called `name` and with value `value`. The value of the constant is accessible by means of the syntax `name()`.

- #define const_ref_nodef(type, name, value)

  Declares a per-class constant of type `type`, called `name` and with value `value`. A constant reference to the constant is accessible by means of the syntax `name()`.

- #define PPL_COMPILE_TIME_CHECK(e, msg) PPL_COMPILE_TIME_CHECK_AUX(e, __LINE__)

  Produces a compilation error if the compile-time constant `e` does not evaluate to `true`

Typedefs

- typedef PPL_COEFFICIENT_TYPE Parma_Polyhedra_Library::Coefficient

  An alias for easily naming the type of PPL coefficients.

- typedef Coefficient_traits_template< Coefficient > Parma_Polyhedra_Library::Coefficient_traits

  An alias for easily naming the coefficient traits.

- typedef size_t Parma_Polyhedra_Library::dimension_type

  An unsigned integral type for representing space dimensions.

- typedef size_t Parma_Polyhedra_Library::memory_size_type

  An unsigned integral type for representing memory size in bytes.

Enumerations

- enum Parma_Polyhedra_Library::Degenerate_Element { Parma_Polyhedra_Library::UNIVERSE, Parma_Polyhedra_Library::EMPTY }

  Kinds of degenerate abstract elements.

- enum Parma_Polyhedra_Library::Relation_Symbol {

  Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN,

  Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::NOT_EQUAL }

Relation symbols.

- enum Parma_Polyhedra_Library::Complexity_Class { Parma_Polyhedra_Library::POLYNOMIAL_COMPLEXITY, Parma_Polyhedra_Library::SIMPLEX_COMPLEXITY, Parma_Polyhedra_Library::ANY_COMPLEXITY }

  Complexity pseudo-classes.

- enum Parma_Polyhedra_Library::Optimization_Mode { Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::MAXIMIZATION }

  Possible optimization modes.

- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Width {
  Parma_Polyhedra_Library::BITS_8, Parma_Polyhedra_Library::BITS_16, Parma_Polyhedra_Library::BITS_32, Parma_Polyhedra_Library::BITS_64,
  Parma_Polyhedra_Library::BITS_128 }

  Widths of bounded integer types.

- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Representation { Parma_Polyhedra_Library::UNSIGNED, Parma_Polyhedra_Library::SIGNED_2_COMPLEMENT }

  Representation of bounded integer types.

- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Overflow { Parma_Polyhedra_Library::OVERFLOW_WRAPS, Parma_Polyhedra_Library::OVERFLOW_UNDEFINED, Parma_Polyhedra_Library::OVERFLOW_IMPOSSIBLE }

  Overflow behavior of bounded integer types.

- enum Parma_Polyhedra_Library::Topology { Parma_Polyhedra_Library::NECESSARILY_CLOSED = 0, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED = 1 }

  Kinds of polyhedra domains.

- enum Parma_Polyhedra_Library::MIP_Problem_Status { Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM, Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM }

  Possible outcomes of the MIP_Problem solver.

- enum Parma_Polyhedra_Library::PIP_Problem_Status { Parma_Polyhedra_Library::UNFEASIBLE_PIP_PROBLEM, Parma_Polyhedra_Library::OPTIMIZED_PIP_PROBLEM }

  Possible outcomes of the PIP_Problem solver.

Variables

- const Throwable ∗volatile Parma_Polyhedra_Library::abandon_expensive_computations = 0

  A pointer to an exception object.

9.1.1 Detailed Description

The core implementation of the Parma Polyhedra Library is written in C++. See Namespace, Hierarchical and Compound indexes for additional information about each single data type.

9.1.2 Define Documentation

9.1.2.1 #define const_bool_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

Declares a per-class constant of type `bool`, called `name` and with value `value`.

Differently from static constants, `name` needs not (and cannot) be defined (for static constants, the need for a further definition is mandated by Section 9.4.2/4 of the C++ standard).

Definition at line 42 of file meta_programming.hh.

9.1.2.2 #define const_int_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

Declares a per-class constant of type `int`, called `name` and with value `value`.

Differently from static constants, `name` needs not (and cannot) be defined (for static constants, the need for a further definition is mandated by Section 9.4.2/4 of the C++ standard).

Definition at line 56 of file meta_programming.hh.

9.1.2.3 #define const_ref_nodef(type, name, value)

Value:

```
static const type& name() {                                          \
    static type name(value);                                         \
    return name;                                                     \
  }
```

Declares a per-class constant of type `type`, called `name` and with value `value`. A constant reference to the constant is accessible by means of the syntax `name()`.

Differently from static constants, `name` needs not (and cannot) be defined (for static constants, the need for a further definition is mandated by Section 9.4.2/4 of the C++ standard).

Definition at line 88 of file meta_programming.hh.

9.1.2.4 #define const_value_nodef(type, name, value)

Value:

```
static type name() {                            \
    return value;                               \
  }
```

Declares a per-class constant of type `type`, called `name` and with value `value`. The value of the constant is accessible by means of the syntax `name()`.

Differently from static constants, `name` needs not (and cannot) be defined (for static constants, the need for a further definition is mandated by Section 9.4.2/4 of the C++ standard).

Definition at line 71 of file meta_programming.hh.

9.1.2.5 #define PPL_COMPILE_TIME_CHECK(e, msg) PPL_COMPILE_TIME_CHECK_AUX(e, __LINE__)

Produces a compilation error if the compile-time constant `e` does not evaluate to `true`

Definition at line 134 of file meta_programming.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::swap(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign().

9.1.2.6 #define PPL_VERSION "0.11.2"

A string containing the PPL version.

Let `M` and `m` denote the numbers associated to PPL_VERSION_MAJOR and PPL_VERSION_MINOR, respectively. The format of PPL_VERSION is `M "." m` if both PPL_VERSION_REVISION (`r`) and PPL_VERSION_BETA (`b`)are zero, `M "." m "pre" b` if PPL_VERSION_REVISION is zero and PPL_VERSION_BETA is not zero, `M "." m "." r` if PPL_VERSION_REVISION is not zero and PPL_VERSION_BETA is zero, `M "." m "." r "pre" b` if neither PPL_VERSION_REVISION nor PPL_VERSION_BETA are zero.

Definition at line 59 of file version.hh.

9.1.2.7 #define PPL_VERSION_BETA 0

The beta number of the PPL version. This is zero for official releases and nonzero for development snapshots.

Definition at line 45 of file version.hh.

Referenced by Parma_Polyhedra_Library::version_beta().

9.1.2.8 #define PPL_VERSION_MAJOR 0

The major number of the PPL version.

Definition at line 30 of file version.hh.

Referenced by Parma_Polyhedra_Library::version_major().

9.1.2.9 #define PPL_VERSION_MINOR 11

The minor number of the PPL version.

Definition at line 34 of file version.hh.

Referenced by Parma_Polyhedra_Library::version_minor().

9.1.2.10 #define PPL_VERSION_REVISION 2

The revision number of the PPL version.

Definition at line 38 of file version.hh.

Referenced by Parma_Polyhedra_Library::version_revision().

9.1.3 Typedef Documentation

9.1.3.1 typedef PPL_COEFFICIENT_TYPE Parma_Polyhedra_Library::Coefficient

An alias for easily naming the type of PPL coefficients.

Objects of type Coefficient are used to implement the integral valued coefficients occurring in linear expressions, constraints, generators, intervals, bounding boxes and so on. Depending on the chosen configuration options (see file `README.configure`), a Coefficient may actually be:

- The GMP_Integer type, which in turn is an alias for the `mpz_class` type implemented by the C++ interface of the GMP library (this is the default configuration).
- An instance of the Checked_Number class template: with the policy Bounded_Integer_Coefficient_Policy, this implements overflow detection on top of a native integral type (available template instances include checked integers having 8, 16, 32 or 64 bits); with the Checked_Number_Transparent_Policy, this is a wrapper for native integral types with no overflow detection (available template instances are as above).

Definition at line 172 of file Coefficient.types.hh.

9.1.3.2 typedef Coefficient_traits_template<Coefficient> Parma_Polyhedra_Library::Coefficient_traits

An alias for easily naming the coefficient traits.

Definition at line 178 of file Coefficient.types.hh.

9.1.3.3 typedef size_t Parma_Polyhedra_Library::dimension_type

An unsigned integral type for representing space dimensions.

Definition at line 22 of file globals.types.hh.

9.1.3.4 typedef size_t Parma_Polyhedra_Library::memory_size_type

An unsigned integral type for representing memory size in bytes.

Definition at line 26 of file globals.types.hh.

9.1.4 Enumeration Type Documentation

9.1.4.1 enum Parma_Polyhedra_Library::Bounded_Integer_Type_Overflow

Overflow behavior of bounded integer types.

See the section on approximating bounded integers.

Enumerator:

OVERFLOW_WRAPS On overflow, wrapping takes place. This means that, for a w-bit bounded integer, the computation happens modulo 2^w.

OVERFLOW_UNDEFINED On overflow, the result is undefined. This simply means that the result of the operation resulting in an overflow can take any value.

Note

Even though something more serious can happen in the system being analyzed ---due to, e.g., C's undefined behavior---, here we are only concerned with the results of arithmetic operations. It is the responsibility of the analyzer to ensure that other manifestations of undefined behavior are conservatively approximated.

OVERFLOW_IMPOSSIBLE Overflow is impossible. This is for the analysis of languages where overflow is trapped before it affects the state, for which, thus, any indication that an overflow may have affected the state is necessarily due to the imprecision of the analysis.

Definition at line 121 of file globals.types.hh.

9.1.4.2 enum Parma_Polyhedra_Library::Bounded_Integer_Type_Representation

Representation of bounded integer types.

See the section on approximating bounded integers.

Enumerator:

UNSIGNED Unsigned binary.

SIGNED_2_COMPLEMENT Signed binary where negative values are represented by the two's complement of the absolute value.

Definition at line 104 of file globals.types.hh.

9.1.4.3 enum Parma_Polyhedra_Library::Bounded_Integer_Type_Width

Widths of bounded integer types.

See the section on approximating bounded integers.

Enumerator:

BITS_8 8 bits.

BITS_16 16 bits.

BITS_32 32 bits.

BITS_64 64 bits.

BITS_128 128 bits.

Definition at line 81 of file globals.types.hh.

9.1.4.4 enum Parma_Polyhedra_Library::Complexity_Class

Complexity pseudo-classes.

Enumerator:

POLYNOMIAL_COMPLEXITY Worst-case polynomial complexity.

SIMPLEX_COMPLEXITY Worst-case exponential complexity but typically polynomial behavior.

ANY_COMPLEXITY Any complexity.

Definition at line 57 of file globals.types.hh.

9.1.4.5 enum Parma_Polyhedra_Library::Degenerate_Element

Kinds of degenerate abstract elements.

Enumerator:

UNIVERSE The universe element, i.e., the whole vector space.

EMPTY The empty element, i.e., the empty set.

Definition at line 30 of file globals.types.hh.

9.1.4.6 enum Parma_Polyhedra_Library::MIP_Problem_Status

Possible outcomes of the MIP_Problem solver.

Enumerator:

UNFEASIBLE_MIP_PROBLEM The problem is unfeasible.

UNBOUNDED_MIP_PROBLEM The problem is unbounded.

OPTIMIZED_MIP_PROBLEM The problem has an optimal solution.

Definition at line 20 of file MIP_Problem.types.hh.

9.1.4.7 enum Parma_Polyhedra_Library::Optimization_Mode

Possible optimization modes.

Enumerator:

MINIMIZATION Minimization is requested.
MAXIMIZATION Maximization is requested.

Definition at line 68 of file globals.types.hh.

9.1.4.8 enum Parma_Polyhedra_Library::PIP_Problem_Status

Possible outcomes of the PIP_Problem solver.

Enumerator:

UNFEASIBLE_PIP_PROBLEM The problem is unfeasible.
OPTIMIZED_PIP_PROBLEM The problem has an optimal solution.

Definition at line 20 of file PIP_Problem.types.hh.

9.1.4.9 enum Parma_Polyhedra_Library::Relation_Symbol

Relation symbols.

Enumerator:

EQUAL Equal to.
LESS_THAN Less than.
LESS_OR_EQUAL Less than or equal to.
GREATER_THAN Greater than.
GREATER_OR_EQUAL Greater than or equal to.
NOT_EQUAL Not equal to.

Definition at line 40 of file globals.types.hh.

9.1.4.10 enum Parma_Polyhedra_Library::Topology

Kinds of polyhedra domains.

Enumerator:

NECESSARILY_CLOSED
NOT_NECESSARILY_CLOSED

Definition at line 22 of file Topology.hh.

9.1.5 Variable Documentation

9.1.5.1 const Throwable ∗volatile Parma_Polyhedra_Library::abandon_expensive_computations = 0

A pointer to an exception object.

This pointer, which is initialized to zero, is repeatedly checked along any super-linear (i.e., computationally expensive) computation path in the library. When it is found nonzero the exception it points to is thrown. In other words, making this pointer point to an exception (and leaving it in this state) ensures that the library will return control to the client application, possibly by throwing the given exception, within a time that is a linear function of the size of the representation of the biggest object (powerset of polyhedra, polyhedron, system of constraints or generators) on which the library is operating upon.

Note

The only sensible way to assign to this pointer is from within a signal handler or from a parallel thread. For this reason, the library, apart from ensuring that the pointer is initially set to zero, never assigns to it. In particular, it does not zero it again when the exception is thrown: it is the client's responsibility to do so.

Definition at line 32 of file globals.cc.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::maybe_abandon(), and Parma_Polyhedra_Library::Powerset< D >::omega_reduce().

10 Namespace Documentation

10.1 Parma_Polyhedra_Library Namespace Reference

The entire library is confined to this namespace.

Namespaces

- namespace Checked

  Types and functions implementing checked numbers.

- namespace Implementation

  Implementation related data and functions.

- namespace Interfaces

  Data and functions related to language interfaces.

- namespace Interval_NS
- namespace IO_Operators

  All input/output operators are confined to this namespace.

Classes

- struct Compile_Time_Check< true >

  A class that is only defined if `b` *evaluates to* `true`.

- struct Bool

  A class holding a constant called `value` *that evaluates to* `b`.

- struct True

  A class holding a constant called `value` *that evaluates to* `true`.

- struct False

  A class holding a constant called `value` *that evaluates to* `false`.

- struct Is_Same

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `T1` *is the same type as* `T2`.

- struct Is_Same< T, T >

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `T1` *is the same type as* `T2`.

- struct Is_Same_Or_Derived

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `Base` *is the same type as* `Derived` *or* `Derived` *is a class derived from* `Base`.

- struct Enable_If

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`.

- struct Enable_If_Is
- struct Enable_If< true, T >

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`.

- struct Is_Native
- struct Is_Native< char >
- struct Is_Native< signed char >
- struct Is_Native< signed short >
- struct Is_Native< signed int >
- struct Is_Native< signed long >
- struct Is_Native< signed long long >
- struct Is_Native< unsigned char >
- struct Is_Native< unsigned short >
- struct Is_Native< unsigned int >
- struct Is_Native< unsigned long >
- struct Is_Native< unsigned long long >
- struct Is_Native< mpz_class >
- struct Is_Native< mpq_class >
- struct float_ieee754_half
- struct float_ieee754_single
- struct float_ieee754_double
- struct float_ibm_single
- struct float_ibm_double

- struct float_intel_double_extended
- struct float_ieee754_quad
- class Float
- struct Extended_Number_Policy
- struct Check_Overflow_Policy

  A policy checking for overflows.

- struct Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Native_Checked_From_Wrapper< Checked_Number< T, P > >
- struct Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Native_Checked_To_Wrapper< Checked_Number< T, P > >
- struct Is_Checked
- struct Is_Checked< Checked_Number< T, P > >
- struct Is_Native_Or_Checked
- class Checked_Number

  A wrapper for numeric types implementing a given policy.

- struct Slow_Copy< Checked_Number< T, P > >
- struct FPU_Related< Checked_Number< T, Policy > >
- struct Coefficient_traits_template< GMP_Integer >

  Coefficient traits specialization for unbounded integers.

- struct Coefficient_traits_template

  Coefficient traits.

- struct Minus_Infinity
- struct Plus_Infinity
- struct Not_A_Number
- struct Is_Special
- struct Is_Special< Minus_Infinity >
- struct Is_Special< Plus_Infinity >
- struct Is_Special< Not_A_Number >
- struct Checked_Number_Transparent_Policy
- class Init

  Class for initialization and finalization.

- struct Weightwatch_Traits

  Traits class for the deterministic timeout mechanism.

- class Throwable

  User objects the PPL can throw.

- struct Recycle_Input

  A tag class.

- struct Fit
- struct Fit< T, v, typename Enable_If< C_Integer< T >::value >::type >
- struct TConstant
- struct Constant_

- struct Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type >
- struct Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type >
- struct Constant
- struct Has_OK
- struct Has_OK< T, typename Enable_If_Is< bool(T::*)() const,&T::OK >::type >
- struct Rectilinear_Distance_Specialization
- struct Euclidean_Distance_Specialization
- struct L_Infinity_Distance_Specialization
- struct maybe_assign_struct
- struct maybe_assign_struct< Type, Type >
- class stdiobuf
- class Row_Impl_Handler

  The handler of the actual Row implementation.

- class Row

  A finite sequence of coefficients.

- class Matrix

  A 2-dimensional matrix of coefficients.

- class Linear_Row

  The base class for linear expressions, constraints and generators.

- class Linear_System

  The base class for systems of constraints and generators.

- class MIP_Problem

  A Mixed Integer (linear) Programming problem.

- class Bit_Row

  A row in a matrix of bits.

- class Bit_Matrix

  A matrix of bits.

- class Variable

  A dimension of the vector space.

- class Variables_Set

  An std::set of variables' indexes.

- class Linear_Expression

  A linear expression.

- class Constraint

  A linear equality or inequality.

- class Congruence

  A linear congruence.

- class Generator

  A line, ray, point or closure point.

- class Grid_Generator

  A grid line, parameter or grid point.

- class Constraint_System

  A system of constraints.

- class Congruence_System

  A system of congruences.

- class Generator_System

  A system of generators.

- class Grid_Generator_System

  A system of grid generators.

- class Scalar_Products

  A class implementing various scalar product functions.

- class Topology_Adjusted_Scalar_Product_Sign

  Scalar product sign function object depending on topology.

- class Polyhedron

  The base class for convex polyhedra.

- class Grid

  A grid.

- class Poly_Con_Relation

  The relation between a polyhedron and a constraint.

- class Poly_Gen_Relation

  The relation between a polyhedron and a generator.

- struct Boundary_Value
- struct Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type >
- class Interval_Restriction_None_Base
- class Interval_Restriction_None
- class Interval_Restriction_Integer_Base
- class Interval_Restriction_Integer
- class Simple_Restriction_Integer
- struct Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >

- struct Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_-Restriction_None_Base, typename From::info_type >::value >::type >
- struct Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_-Restriction_Integer_Base, typename From::info_type >::value >::type >
- class Interval_Restriction_Integer_Modulo_Base
- class Interval_Restriction_Integer_Modulo
- struct Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > >
- struct Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_-Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >
- struct Simple_Restriction_Integer_Modulo
- struct Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >
- struct Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >
- struct Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >
- struct Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >
- class Interval_Info_Null
- class Interval_Info_Null_Open
- class Interval_Info_Bitset
- struct Interval_Base
- struct Is_Singleton
- struct Is_Interval
- class Interval

  A generic, not necessarily closed, possibly restricted interval.

- struct Select_Temp_Boundary_Type
- struct Select_Temp_Boundary_Type< float >
- struct Select_Temp_Boundary_Type< char >
- struct Select_Temp_Boundary_Type< signed char >
- struct Select_Temp_Boundary_Type< unsigned char >
- struct Select_Temp_Boundary_Type< signed short >
- struct Select_Temp_Boundary_Type< unsigned short >
- struct Select_Temp_Boundary_Type< signed int >
- struct Select_Temp_Boundary_Type< unsigned int >
- struct Select_Temp_Boundary_Type< signed long >
- struct Select_Temp_Boundary_Type< unsigned long >
- struct Select_Temp_Boundary_Type< unsigned long long >
- class C_Polyhedron

  A closed convex polyhedron.

- class NNC_Polyhedron

  A not necessarily closed convex polyhedron.

- class Widening_Function

  Wraps a widening method into a function object.

- class Limited_Widening_Function

  Wraps a limited widening method into a function object.

- class BHRZ03_Certificate

  The convergence certificate for the BHRZ03 widening operator.

- class H79_Certificate

  A convergence certificate for the H79 widening operator.

- class Grid_Certificate

  The convergence certificate for the Grid widening operator.

- class Determinate

  A wrapper for PPL pointsets, providing them with a determinate constraint system interface, as defined in [Bag98].

- class iterator_to_const

  An iterator on a sequence of read-only objects.

- class const_iterator_to_const

  A const_iterator on a sequence of read-only objects.

- class Powerset

  The powerset construction on a base-level domain.

- class Pointset_Powerset

  The powerset construction instantiated on PPL pointset domains.

- class Smash_Reduction

  This class provides the reduction method for the Smash_Product domain.

- class Constraints_Reduction

  This class provides the reduction method for the Constraints_Product domain.

- class Congruences_Reduction

  This class provides the reduction method for the Congruences_Product domain.

- class Shape_Preserving_Reduction

  This class provides the reduction method for the Shape_Preserving_Product domain.

- class No_Reduction

  This class provides the reduction method for the Direct_Product domain.

- class Partially_Reduced_Product

  The partially reduced product of two abstractions.

- class Domain_Product

  This class is temporary and will be removed when template typedefs will be supported in C++.

- class Box

  A not necessarily closed, iso-oriented hyperrectangle.

- class DB_Row_Impl_Handler

  The handler of the actual DB_Row implementation.

- class DB_Row

  The base class for the single rows of matrices.

- class DB_Matrix

  The base class for the square matrices.

- class BD_Shape

  A bounded difference shape.

- class OR_Matrix

  A matrix representing octagonal constraints.

- class Octagonal_Shape

  An octagonal shape.

- class PIP_Problem

  A Parametric Integer (linear) Programming problem.

- class PIP_Tree_Node

  A node of the PIP solution tree.

- class PIP_Solution_Node

  A tree node representing part of the space of solutions.

- class PIP_Decision_Node

  A tree node representing a decision in the space of solutions.

- class GMP_Integer

  Unbounded integers as provided by the GMP library.

Typedefs

- typedef mpz_class GMP_Integer
- typedef PPL_COEFFICIENT_TYPE Coefficient

  An alias for easily naming the type of PPL coefficients.

- typedef Coefficient_traits_template< Coefficient > Coefficient_traits

  An alias for easily naming the coefficient traits.

- typedef size_t dimension_type

  An unsigned integral type for representing space dimensions.

- typedef size_t memory_size_type

  An unsigned integral type for representing memory size in bytes.

- typedef const PIP_Tree_Node * PIP_Tree

Enumerations

- enum Degenerate_Element { UNIVERSE, EMPTY }

  Kinds of degenerate abstract elements.

- enum Relation_Symbol {
  EQUAL, LESS_THAN, LESS_OR_EQUAL, GREATER_THAN,
  GREATER_OR_EQUAL, NOT_EQUAL }

  Relation symbols.

- enum Complexity_Class { POLYNOMIAL_COMPLEXITY, SIMPLEX_COMPLEXITY, ANY_-COMPLEXITY }

  Complexity pseudo-classes.

- enum Optimization_Mode { MINIMIZATION, MAXIMIZATION }

  Possible optimization modes.

- enum Bounded_Integer_Type_Width {
  BITS_8, BITS_16, BITS_32, BITS_64,
  BITS_128 }

  Widths of bounded integer types.

- enum Bounded_Integer_Type_Representation { UNSIGNED, SIGNED_2_COMPLEMENT }

  Representation of bounded integer types.

- enum Bounded_Integer_Type_Overflow { OVERFLOW_WRAPS, OVERFLOW_UNDEFINED, OVERFLOW_IMPOSSIBLE }

  Overflow behavior of bounded integer types.

- enum Topology { NECESSARILY_CLOSED = 0, NOT_NECESSARILY_CLOSED = 1 }

  Kinds of polyhedra domains.

- enum MIP_Problem_Status { UNFEASIBLE_MIP_PROBLEM, UNBOUNDED_MIP_PROBLEM, OPTIMIZED_MIP_PROBLEM }

  Possible outcomes of the MIP_Problem solver.

- enum Ternary { T_YES, T_NO, T_MAYBE }
- enum PIP_Problem_Status { UNFEASIBLE_PIP_PROBLEM, OPTIMIZED_PIP_PROBLEM }

  Possible outcomes of the PIP_Problem solver.

Functions

- unsigned version_major ()

  Returns the major number of the PPL version.

- unsigned version_minor ()

  Returns the minor number of the PPL version.

- unsigned version_revision ()

  Returns the revision number of the PPL version.

- unsigned version_beta ()

  Returns the beta number of the PPL version.

- const char * version ()

  Returns a character string containing the PPL version.

- const char * banner ()

  Returns a character string containing the PPL banner.

- void throw_result_exception (Result r)
- template<typename T >
  T plus_infinity ()
- template<typename T >
  T minus_infinity ()
- template<typename T >
  T not_a_number ()
- template<typename T >
  void maybe_reset_fpu_inexact ()
- template<typename T >
  int maybe_check_fpu_inexact ()
- Rounding_Dir rounding_dir (Rounding_Dir dir)
- Result check_result (Result r, Rounding_Dir dir)
- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type assign_r (To &to, const From &, Rounding_Dir dir)
- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type construct (To &to, const From &, Rounding_Dir dir)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_minus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_plus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, int >::type is_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_not_a_number (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_integer (const T &x)
- template<typename T , typename Policy >
  bool is_not_a_number (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  bool is_minus_infinity (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  bool is_plus_infinity (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  void exact_div_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
- void neg_assign (GMP_Integer &x)

- void neg_assign (GMP_Integer &x, const GMP_Integer &y)
- void abs_assign (GMP_Integer &x)
- void abs_assign (GMP_Integer &x, const GMP_Integer &y)
- void gcd_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void rem_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void gcdext_assign (GMP_Integer &x, GMP_Integer &s, GMP_Integer &t, const GMP_Integer &y, const GMP_Integer &z)
- void lcm_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void add_mul_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void sub_mul_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void mul_2exp_assign (GMP_Integer &x, const GMP_Integer &y, unsigned int exp)
- void div_2exp_assign (GMP_Integer &x, const GMP_Integer &y, unsigned int exp)
- void exact_div_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
- void sqrt_assign (GMP_Integer &x, const GMP_Integer &y)
- int cmp (const GMP_Integer &x, const GMP_Integer &y)
- const mpz_class & raw_value (const GMP_Integer &x)
- mpz_class & raw_value (GMP_Integer &x)
- void Coefficient_constants_initialize ()

  Initializes the Coefficient constants.

- void Coefficient_constants_finalize ()

  Finalizes the Coefficient constants.

- Coefficient_traits::const_reference Coefficient_zero ()

  Returns a const reference to a Coefficient with value 0.

- Coefficient_traits::const_reference Coefficient_one ()

  Returns a const reference to a Coefficient with value 1.

- unsigned irrational_precision ()

  Returns the precision parameter used for irrational calculations.

- void set_irrational_precision (const unsigned p)

  Sets the precision parameter used for irrational calculations.

- void set_rounding_for_PPL ()

  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

- void restore_pre_PPL_rounding ()

  Sets the FPU rounding mode as it was before initialization of the PPL.

- void initialize ()

  Initializes the library.

- void finalize ()

  Finalizes the library.

- dimension_type not_a_dimension ()

  Returns a value that does not designate a valid dimension.

- template<typename T >
  Enable_If< Slow_Copy< T >::value, void >::type swap (T &, T &)

  Make sure swap() is specialized when needed.

- dimension_type compute_capacity (dimension_type requested_size, dimension_type maximum_size)

  Speculative allocation function.

- template<typename T >
  Enable_If< Has_OK< T >::value, bool >::type f_OK (const T &to)
- void maybe_abandon ()
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type numer_denom (const T &from, Coefficient &num, Coefficient &den)

  Extract the numerator and denominator components of `from`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type div_round_up (T &to, Coefficient_traits::const_reference x, Coefficient_traits::const_reference y)

  Divides `x` by `y` into `to`, rounding the result towards plus infinity.

- template<typename N >
  void min_assign (N &x, const N &y)

  Assigns to `x` the minimum between `x` and `y`.

- template<typename N >
  void max_assign (N &x, const N &y)

  Assigns to `x` the maximum between `x` and `y`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_even (const T &x)

  Returns `true` if and only if `x` is an even number.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_additive_inverse (const T &x, const T &y)

  Returns `true` if and only if $x = -y$.

- void normalize2 (Coefficient_traits::const_reference x, Coefficient_traits::const_reference y, Coefficient &nx, Coefficient &ny)

  If `g` is the GCD of `x` and `y`, the values of `x` and `y` divided by `g` are assigned to `nx` and `ny`, respectively.

- bool is_canonical (const mpq_class &x)

  Returns `true` if and only if `x` is in canonical form.

- template<typename T >
  T low_bits_mask (unsigned n)

  Returns a mask for the lowest `n` bits,.

- template<typename To , typename From >
  Result maybe_assign (const To *&top, To &tmp, const From &from, Rounding_Dir dir)

Assigns to `top` a pointer to a location that holds the conversion, according to `dir`, of `from` to type `To`. When necessary, and only when necessary, the variable `tmp` is used to hold the result of conversion.

- template<typename PSET >
  bool termination_test_MS (const PSET &pset)

  Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool termination_test_MS_2 (const PSET &pset_before, const PSET &pset_after)

  Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool one_affine_ranking_function_MS (const PSET &pset, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool one_affine_ranking_function_MS_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void all_affine_ranking_functions_MS (const PSET &pset, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void all_affine_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void all_affine_quasi_ranking_functions_MS (const PSET &pset, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void all_affine_quasi_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool termination_test_PR (const PSET &pset)

  Like termination_test_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool termination_test_PR_2 (const PSET &pset_before, const PSET &pset_after)

Like termination_test_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool one_affine_ranking_function_PR (const PSET &pset, Generator &mu)

  Like one_affine_ranking_function_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool one_affine_ranking_function_PR_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

  Like one_affine_ranking_function_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  void all_affine_ranking_functions_PR (const PSET &pset, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  void all_affine_ranking_functions_PR_2 (const PSET &pset_before, const PSET &pset_after, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- bool eq_restriction (const Interval_Restriction_None_Base &, const Interval_Restriction_None_Base)
- template<typename T >
  bool contains_restriction (const Interval_Restriction_None_Base &, const T &)
- template<typename T >
  bool assign_restriction (Interval_Restriction_None_Base &, const T &)
- template<typename T1 , typename T2 >
  bool join_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool intersect_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool diff_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T >
  bool neg_restriction (Interval_Restriction_None_Base &, const T &)
- template<typename T1 , typename T2 >
  bool add_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool sub_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool mul_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool div_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- void output_restriction (std::ostream &, const Interval_Restriction_None_Base &)
- template<typename T1 , typename T2 >
  Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T1 >::value &&Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T2 >::value, bool >::type eq_restriction (const T1 &x, const T2 &y)

- template<typename T1 , typename T2 >
  Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T1 >::value &&Is_Same_-Or_Derived< Interval_Restriction_Integer_Base, T2 >::value, bool >::type contains_restriction (const T1 &x, const T2 &y)
- template<typename Base , typename From >
  bool assign_restriction (Interval_Restriction_Integer< Base > &to, const From &x)
- template<typename Base , typename From1 , typename From2 >
  bool join_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool intersect_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool diff_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &)
- template<typename Base , typename From >
  bool neg_restriction (Interval_Restriction_Integer< Base > &to, const From &x)
- template<typename Base , typename From1 , typename From2 >
  bool add_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool sub_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool mul_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool div_restriction (Interval_Restriction_Integer< Base > &to, const From1 &, const From2 &)
- template<typename Base >
  void output_restriction (std::ostream &s, const Interval_Restriction_Integer< Base > &x)
- template<typename T , typename Base >
  bool set_unrestricted (Interval_Restriction_Integer_Modulo< T, Base > &to)
- template<typename T , typename Base >
  bool set_integer (Interval_Restriction_Integer_Modulo< T, Base > &to)
- template<typename T , typename Base , typename From >
  bool assign_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From &x)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool join_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool intersect_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool diff_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From >
  bool neg_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From &x)
- template<typename T >
  void addmod (T &to, const T &x, const T &y, const T &to_m, const T &y_m)
- template<typename M , typename T >
  bool assign_rem (M &rem, const T &n, const M &div)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool add_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T >
  void submod (T &to, const T &x, const T &y, const T &to_m, const T &y_m)

- template<typename T , typename Base , typename From1 , typename From2 >
  bool sub_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T >
  void mulmod (T &to, const T &x, const T &y, const T &to_m)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool mul_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool div_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base >
  void output_restriction (std::ostream &s, const Interval_Restriction_Integer_Modulo< T, Base > &x)
- I_Result combine (Result l, Result u)
- template<typename From >
  Enable_If< Is_Interval< From >::value, I_Result >::type neg_assign (From &x)
- template<typename Boundary , typename Info >
  bool f_is_empty (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  bool f_is_singleton (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  int is_infinity (const Interval< Boundary, Info > &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type is_singleton_integer (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type check_empty_arg (const T &x)
- template<typename T1 , typename T2 >
  Enable_If<((Is_Singleton< T1 >::value||Is_Interval< T1 >::value)&&(Is_Singleton< T2 >::value||Is_Interval< T2 >::value)&&(Is_Interval< T1 >::value||Is_Interval< T2 >::value)), bool >::type operator== (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<((Is_Singleton< T1 >::value||Is_Interval< T1 >::value)&&(Is_Singleton< T2 >::value||Is_Interval< T2 >::value)&&(Is_Interval< T1 >::value||Is_Interval< T2 >::value)), bool >::type operator!= (const T1 &x, const T2 &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator+ (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator+ (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > operator+ (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator- (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator- (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > operator- (const Interval< B, Info > &x, const Interval< B, Info > &y)

- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator∗ (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator∗ (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > operator∗ (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator/ (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type operator/ (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > operator/ (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename Boundary , typename Info >
  std::ostream & operator<< (std::ostream &os, const Interval< Boundary, Info > &x)
- template<typename D1 , typename D2 >
  bool shrink_to_congruence_no_check (D1 &d1, D2 &d2, const Congruence &cg)
- template<typename ITV >
  Poly_Con_Relation interval_relation (const ITV &i, const Constraint::Type constraint_type, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den=1)

  Returns the relations holding between an interval and an interval constraint.

- bool extract_interval_congruence (const Congruence &cg, dimension_type cg_space_dim, dimension_type &cg_num_vars, dimension_type &cg_only_var)
- template<typename ITV >
  bool operator!= (const Box< ITV > &x, const Box< ITV > &y)
- template<typename ITV >
  bool operator== (const Box< ITV > &x, const Box< ITV > &y)
- unsigned long isqrt (unsigned long x)

  Returns the integer square root of x.

Relational Operators and Comparison Functions

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type not_equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type greater_or_equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type greater_than (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type less_or_equal (const T1 &x, const T2 &y)

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type less_than (const T1 &x, const T2 &y)

Input-Output Operators

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type ascii_dump (std::ostream &s, const T &t)

  Ascii dump for native or checked.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type ascii_load (std::ostream &s, T &t)

  Ascii load for native or checked.

Memory Size Inspection Functions

- memory_size_type total_memory_in_bytes (const GMP_Integer &x)
- memory_size_type external_memory_in_bytes (const GMP_Integer &x)
- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type total_memory_in_bytes (const T &)

  For native types, returns the total size in bytes of the memory occupied by the type of the (unused) parameter, i.e., 0.

- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type external_memory_in_bytes (const T &)

  For native types, returns the size in bytes of the memory managed by the type of the (unused) parameter, i.e., 0.

- memory_size_type total_memory_in_bytes (const mpz_class &x)

  Returns the total size in bytes of the memory occupied by x.

- memory_size_type external_memory_in_bytes (const mpz_class &x)

  Returns the size in bytes of the memory managed by x.

- memory_size_type total_memory_in_bytes (const mpq_class &x)

  Returns the total size in bytes of the memory occupied by x.

- memory_size_type external_memory_in_bytes (const mpq_class &x)

  Returns the size in bytes of the memory managed by x.

Variables

- Minus_Infinity MINUS_INFINITY
- Plus_Infinity PLUS_INFINITY
- Not_A_Number NOT_A_NUMBER
- const Throwable *volatile abandon_expensive_computations = 0

  A pointer to an exception object.

10.1.1 Detailed Description

The entire library is confined to this namespace.

10.1.2 Typedef Documentation

10.1.2.1 typedef mpz_class Parma_Polyhedra_Library::GMP_Integer

Definition at line 31 of file GMP_Integer.types.hh.

10.1.2.2 typedef const PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Tree

Definition at line 20 of file PIP_Tree.types.hh.

10.1.3 Enumeration Type Documentation

10.1.3.1 enum Parma_Polyhedra_Library::Ternary

Enumerator:

T_YES

T_NO

T_MAYBE

Definition at line 40 of file Interval.defs.hh.

10.1.4 Function Documentation

10.1.4.1 void Parma_Polyhedra_Library::abs_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Definition at line 47 of file GMP_Integer.inlines.hh.

10.1.4.2 void Parma_Polyhedra_Library::abs_assign (GMP_Integer & *x*) [related]

Definition at line 42 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), and Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index().

10.1.4.3 void Parma_Polyhedra_Library::add_mul_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 75 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Grid::reduce_pc_with_pc(), Parma_Polyhedra_Library::Scalar_Products::reduced_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index(), Parma_Polyhedra_Library::PIP_Solution_Node::update_solution(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

10.1.4.4 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 679 of file Interval_Restriction.defs.hh.

References addmod(), assign_r(), assign_rem(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder, Parma_Polyhedra_Library::Checked::Result, set_integer(), and set_unrestricted().

10.1.4.5 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 279 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.6 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 104 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign().

10.1.4.7 template<typename T > void Parma_Polyhedra_Library::addmod (T & *to*, const T & *x*, const T & *y*, const T & *to_m*, const T & *y_m*) [inline]

Definition at line 633 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Checked::Result.

Referenced by add_restriction().

10.1.4.8 template<typename PSET > void Parma_Polyhedra_Library::all_affine_quasi_-ranking_functions_MS (const PSET & *pset*, C_Polyhedron & *decreasing_mu_space*, C_Polyhedron & *bounded_mu_space*) [inline]

Computes the spaces of affine *quasi* ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset A pointset approximating the behavior of a loop whose termination is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

where unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update.

decreasing_mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the decreasing affine functions for the loops that are precisely characterized by `pset`.

bounded_mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the lower bounded affine functions for the loops that are precisely characterized by `pset`.

These quasi-ranking functions are of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ identify any point of the `decreasing_mu_space` and `bounded_mu_space` polyhedrons. The variables $\mu_0, \mu_1, \ldots, \mu_n$ correspond to the space dimensions $n, 0, \ldots, n-1$, respectively. When `decreasing_mu_space` (resp., `bounded_mu_space`) is empty, it means that the test is inconclusive. However, if `pset` *precisely* characterizes the effect of the loop body onto the loop-relevant program variables, then `decreasing_-mu_space` (resp., `bounded_mu_space`) will be empty *if and only if* there is no decreasing (resp., lower bounded) affine function, so that the loop does not terminate.

Definition at line 350 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_-approximation(), and UNIVERSE.

Referenced by all_affine_quasi_ranking_functions_MS_2().

10.1.4.9 template<typename PSET > void Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS_2 (const PSET & *pset_before*, const PSET & *pset_after*, C_Polyhedron & *decreasing_mu_space*, C_Polyhedron & *bounded_mu_space*) [inline]

Computes the spaces of affine *quasi* ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset_before A pointset approximating the values of loop-relevant variables *before* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x_1, \ldots, x_n$ go onto space dimensions $0, \ldots, n-1$.

pset_after A pointset approximating the values of loop-relevant variables *after* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

Note that unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update. Note also that unprimed variables are assigned to different space dimensions in `pset_before` and `pset_after`.

Parameters

decreasing_mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the decreasing affine functions for the loops that are precisely characterized by `pset`.

bounded_mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the lower bounded affine functions for the loops that are precisely characterized by `pset`.

These ranking functions are of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ identify any point of the `decreasing_mu_space` and `bounded_mu_space` polyhedrons. The variables $\mu_0, \mu_1, \ldots, \mu_n$ correspond to the space dimensions $n, 0, \ldots, n-1$, respectively. When `decreasing_mu_space` (resp., `bounded_mu_space`) is empty, it means that the test is inconclusive. However, if `pset_before` and `pset_after` *precisely* characterize the effect of the loop body onto the loop-relevant program variables, then `decreasing_mu_space` (resp., `bounded_mu_space`) will be empty *if and only if* there is no decreasing (resp., lower bounded) affine function, so that the loop does not terminate.

Definition at line 379 of file termination.templates.hh.

References all_affine_quasi_ranking_functions_MS(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and UNIVERSE.

10.1.4.10 template<typename PSET > void Parma_Polyhedra_Library::all_affine_ranking_functions_MS (const PSET & *pset*, C_Polyhedron & *mu_space*) [inline]

Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset A pointset approximating the behavior of a loop whose termination is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

where unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update.

mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the affine ranking functions for the loops that are precisely characterized by `pset`. These ranking functions are of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ identify any point of the `mu_space` polyhedron. The variables $\mu_0, \mu_1, \ldots, \mu_n$ correspond to the space dimensions of `mu_space` $n, 0, \ldots, n-1$, respectively. When `mu_space` is empty, it means that the test is inconclusive. However, if `pset` *precisely* characterizes the effect of the loop body onto the loop-relevant program variables, then `mu_space` is empty *if and only if* the loop does *not* terminate.

Definition at line 299 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and UNIVERSE.

Referenced by all_affine_ranking_functions_MS_2().

10.1.4.11 template<typename PSET > void Parma_Polyhedra_Library::all_affine_ranking_functions_MS_2 (const PSET & *pset_before*, const PSET & *pset_after*, C_Polyhedron & *mu_space*) [inline]

Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset_before A pointset approximating the values of loop-relevant variables *before* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x_1, \ldots, x_n$ go onto space dimensions $0, \ldots, n-1$.

pset_after A pointset approximating the values of loop-relevant variables *after* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

Note that unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update. Note also that unprimed variables are assigned to different space dimensions in `pset_before` and `pset_after`.

Parameters

mu_space This is assigned a closed polyhedron of space dimension $n+1$ representing the space of all the affine ranking functions for the loops that are precisely characterized by `pset`. These ranking functions are of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ identify any point of the `mu_space` polyhedron. The variables $\mu_0, \mu_1, \ldots, \mu_n$ correspond to the space dimensions of `mu_space` $n, 0, \ldots, n-1$, respectively. When `mu_space` is empty, it means that the test is inconclusive. However, if `pset_before` and `pset_after` *precisely* characterize the effect of the loop body onto the loop-relevant program variables, then `mu_space` is empty *if and only if* the loop does *not* terminate.

Definition at line 322 of file termination.templates.hh.

References all_affine_ranking_functions_MS(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and UNIVERSE.

10.1.4.12 template<typename PSET > void Parma_Polyhedra_Library::all_affine_ranking_functions_PR (const PSET & *pset*, NNC_Polyhedron & *mu_space*) [inline]

Like all_affine_ranking_functions_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 524 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR_original(), and Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation().

Referenced by all_affine_ranking_functions_PR_2().

10.1.4.13 template<typename PSET > void Parma_Polyhedra_Library::all_affine_ranking_functions_PR_2 (const PSET & *pset_before*, const PSET & *pset_after*, NNC_Polyhedron & *mu_space*) [inline]

Like all_affine_ranking_functions_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 494 of file termination.templates.hh.

References all_affine_ranking_functions_PR(), and Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation().

10.1.4.14 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, void>::type Parma_Polyhedra_Library::ascii_dump (std::ostream & *s*, const T & *t*) [inline]

Ascii dump for native or checked.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_dump().

10.1.4.15 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, bool>::type Parma_Polyhedra_Library::ascii_load (std::ostream & *s*, T & *t*) [inline]

Ascii load for native or checked.

Referenced by Parma_Polyhedra_Library::Matrix::ascii_load(), Parma_Polyhedra_Library::Linear_System::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load(), and Parma_Polyhedra_Library::Congruence_System::ascii_load().

10.1.4.16 template<typename To , typename From > Enable_If<Is_Native_Or_Checked<To>::value && Is_Special<From>::value, Result>::type Parma_Polyhedra_Library::assign_r (To & *to*, const From &, Rounding_Dir *dir*) [related]

Definition at line 205 of file Checked_Number.inlines.hh.

References check_result(), raw_value(), and rounding_dir().

Referenced by add_restriction(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), assign_rem(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::construct_upward_approximation(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), div_round_up(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_binary_octagonal_-

constraints(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::maybe_assign_struct< To, From >::function(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), interval_relation(), Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), join_restriction(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Generator::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), mul_restriction(), mulmod(), numer_denom(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Box< ITV >::refine_interval_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::restrict(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::MIP_Problem::solve_mip(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), sub_restriction(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

10.1.4.17 template<typename M , typename T > bool Parma_Polyhedra_Library::assign_rem (M & *rem*, const T & *n*, const M & *div*) [inline]

Definition at line 657 of file Interval_Restriction.defs.hh.

References assign_r(), and Parma_Polyhedra_Library::Checked::Result.

Referenced by add_restriction(), and sub_restriction().

10.1.4.18 template<typename T , typename Base , typename From > bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From & *x*) [inline]

Definition at line 520 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

10.1.4.19 template<typename Base , typename From > bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_Integer< Base > & *to*, const From & *x*) [inline]

Definition at line 241 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.20 template<typename T > bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_None_Base &, const T &) [inline]

Definition at line 74 of file Interval_Restriction.defs.hh.

Referenced by neg_restriction().

10.1.4.21 const char * Parma_Polyhedra_Library::banner ()

Returns a character string containing the PPL banner.

The banner provides information about the PPL version, the licensing, the lack of any warranty whatsoever, the C++ compiler used to build the library, where to report bugs and where to look for further information.

Definition at line 104 of file version.cc.

10.1.4.22 template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, bool>::type Parma_Polyhedra_Library::check_empty_arg (const T & *x*) [inline]

Definition at line 150 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval_NS::f_info(), and f_is_empty().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), operator<<(), operator==(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.23 Result Parma_Polyhedra_Library::check_result (Result *r*, Rounding_Dir *dir*) [inline]

Definition at line 48 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked::result_relation().

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::assign_r(), assign_r(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number(), construct(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::input(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::output().

10.1.4.24 int Parma_Polyhedra_Library::cmp (const GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Definition at line 106 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Checked::cmp_mp(), Parma_Polyhedra_Library::Linear_Row::compare(), and Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates().

10.1.4.25 void Parma_Polyhedra_Library::Coefficient_constants_finalize ()

Finalizes the Coefficient constants.

Referenced by Parma_Polyhedra_Library::Init::∼Init().

10.1.4.26 void Parma_Polyhedra_Library::Coefficient_constants_initialize ()

Initializes the Coefficient constants.

Referenced by Parma_Polyhedra_Library::Init::Init().

10.1.4.27 Coefficient_traits::const_reference Parma_Polyhedra_Library::Coefficient_one ()

Returns a const reference to a Coefficient with value 1.

Referenced by Parma_Polyhedra_Library::Constraint::initialize(), Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

10.1.4.28 Coefficient_traits::const_reference Parma_Polyhedra_Library::Coefficient_zero ()

Returns a const reference to a Coefficient with value 0.

Referenced by Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Linear_Expression::initialize(), Parma_Polyhedra_Library::Linear_Expression::operator<<(), Parma_Polyhedra_Library::Constraint::operator<<(), Parma_Polyhedra_Library::Congruence::operator<<(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

10.1.4.29 I_Result Parma_Polyhedra_Library::combine (Result *l*, Result *u*) [inline]

Definition at line 43 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Generator::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.30 dimension_type Parma_Polyhedra_Library::compute_capacity (dimension_type *requested_size*, dimension_type *maximum_size*) [inline]

Speculative allocation function.

Returns

The actual capacity to be allocated.

Parameters

requested_size The number of elements we need.

maximum_size The maximum number of elements to be allocated. It is assumed to be no less than `requested_size`.

Computes a capacity given a requested size. Allows for speculative allocation aimed at reducing the number of reallocations enough to guarantee amortized constant insertion time for our vector-like data structures. In all cases, the speculative allocation will not exceed `maximum_size`.

Definition at line 69 of file globals.inlines.hh.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Matrix::add_recycled_row(), Parma_Polyhedra_Library::Bit_Matrix::add_recycled_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::DB_Row< T >::DB_Row(), Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::operator=(), Parma_Polyhedra_Library::Matrix::operator=(), Parma_Polyhedra_Library::DB_Matrix< T >::operator=(), Parma_Polyhedra_Library::Bit_Matrix::resize(), Parma_Polyhedra_Library::Matrix::resize_no_copy(), Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), Parma_Polyhedra_Library::Row::Row(), and Parma_Polyhedra_Library::Polyhedron::simplify().

10.1.4.31 template<typename To , typename From > Enable_If<Is_Native_Or_Checked<To>::value && Is_Special<From>::value, Result>::type Parma_Polyhedra_Library::construct (To & *to*, const From &, Rounding_Dir *dir*) [related]

Definition at line 215 of file Checked_Number.inlines.hh.

References check_result(), raw_value(), and rounding_dir().

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::construct_upward_approximation(), Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix(), Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::Matrix::Matrix(), Parma_Polyhedra_Library::Matrix::resize_no_copy(), and Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy().

10.1.4.32 template<typename T1 , typename T2 > Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, T1 >::value &&Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, T2 >::value, bool >::type Parma_Polyhedra_Library::contains_restriction (const T1 & *x*, const T2 & *y*) [inline]

Definition at line 484 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Checked::Result.

10.1.4.33 template<typename T > bool Parma_Polyhedra_Library::contains_restriction (const Interval_Restriction_None_Base &, const T &) [inline]

Definition at line 68 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), join_restriction(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains().

10.1.4.34 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 618 of file Interval_Restriction.defs.hh.

10.1.4.35 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 &) [inline]

Definition at line 264 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.36 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 92 of file Interval_Restriction.defs.hh.

10.1.4.37 void Parma_Polyhedra_Library::div_2exp_assign (GMP_Integer & *x*, const GMP_Integer & *y*, unsigned int *exp*) [related]

Definition at line 90 of file GMP_Integer.inlines.hh.

10.1.4.38 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 853 of file Interval_Restriction.defs.hh.

References set_unrestricted().

10.1.4.39 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 &, const From2 &) [inline]

Definition at line 303 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.40 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 122 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign().

10.1.4.41 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::div_round_up (T & *to*, Coefficient_traits::const_reference *x*, Coefficient_traits::const_reference *y*) [inline]

Divides `x` by `y` into `to`, rounding the result towards plus infinity.

Definition at line 65 of file math_utilities.inlines.hh.

References assign_r().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), and Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check().

10.1.4.42 template<typename T1 , typename T2 > Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, T1 >::value &&Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, T2 >::value, bool >::type Parma_Polyhedra_Library::eq_restriction (const T1 & *x*, const T2 & *y*) [inline]

Definition at line 477 of file Interval_Restriction.defs.hh.

10.1.4.43 bool Parma_Polyhedra_Library::eq_restriction (const Interval_Restriction_None_Base &, const *Interval_Restriction_None_Base*) [inline]

Definition at line 62 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), operator==(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains().

10.1.4.44 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::equal (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.45 void Parma_Polyhedra_Library::exact_div_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 95 of file GMP_Integer.inlines.hh.

10.1.4.46 template<typename T , typename Policy > void Parma_Polyhedra_Library::exact_-div_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Definition at line 710 of file Checked_Number.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_-Parameter(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_-Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_-Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_-index(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Generator::is_matching_-closure_point(), Parma_Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau::normalize(), normalize2(), Parma_Polyhedra_Library::Congruence_System::normalize_-moduli(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_-Library::Grid::reduce_equality_with_equality(), Parma_Polyhedra_Library::Grid::reduce_line_-with_line(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_-Library::Grid::reduce_pc_with_pc(), Parma_Polyhedra_Library::Grid_Generator::scale_to_divisor(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::MIP_-Problem::steepest_edge_exact_entering_index().

10.1.4.47 memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpq_class & *x*) [inline]

Returns the size in bytes of the memory managed by `x`.

Definition at line 107 of file globals.inlines.hh.

References external_memory_in_bytes().

10.1.4.48 memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpz_class & *x*) [inline]

Returns the size in bytes of the memory managed by `x`.

Definition at line 97 of file globals.inlines.hh.

10.1.4.49 template<typename T > Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_Library::external_memory_in_bytes (const T &) [inline]

For native types, returns the size in bytes of the memory managed by the type of the (unused) parameter, i.e., 0.

Definition at line 85 of file globals.inlines.hh.

10.1.4.50 memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const GMP_Integer & *x*)

Referenced by Parma_Polyhedra_Library::Row_Impl_Handler::Impl::external_memory_in_bytes(), external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::external_memory_in_bytes(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::external_memory_in_bytes(), and total_memory_in_bytes().

10.1.4.51 bool Parma_Polyhedra_Library::extract_interval_congruence (const Congruence & *cg*, dimension_type *cg_space_dim*, dimension_type & *cg_num_vars*, dimension_type & *cg_only_var*)

Definition at line 55 of file Box.cc.

References Parma_Polyhedra_Library::Congruence::coefficient(), Parma_Polyhedra_Library::Congruence::is_equality(), and Parma_Polyhedra_Library::Congruence::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check().

10.1.4.52 template<typename Boundary , typename Info > bool Parma_Polyhedra_Library::f_is_empty (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 63 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty().

Referenced by check_empty_arg().

10.1.4.53 template<typename Boundary , typename Info > bool Parma_Polyhedra_Library::f_is_singleton (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 68 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_singleton().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

10.1.4.54 template<typename T > Enable_If<Has_OK<T>::value, bool>::type Parma_Polyhedra_Library::f_OK (const T & *to*) [inline]

Definition at line 444 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info

>::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_-assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::neg_assign(), operator==(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_-universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), and Parma_-Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.55 void Parma_Polyhedra_Library::finalize () [inline]

Finalizes the library.

Definition at line 56 of file initializer.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_-Library::Generator::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_-distance_assign(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), and Parma_-Polyhedra_Library::Init::~Init().

10.1.4.56 void Parma_Polyhedra_Library::gcd_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 52 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::MIP_Problem::choose_branching_variable(), Parma_Polyhedra_-Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::Polyhedron::contains_-integer_point(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_-Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Grid::frequency_-no_check(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Generator::is_-matching_closure_point(), Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable(), Parma_-Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_-Library::PIP_Solution_Node::Tableau::normalize(), normalize2(), Parma_Polyhedra_Library::MIP_-Problem::OK(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_-Polyhedra_Library::Grid::reduce_equality_with_equality(), Parma_Polyhedra_Library::Grid::reduce_-line_with_line(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_-Library::Grid::relation_with(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_-Polyhedra_Library::MIP_Problem::solve_mip().

10.1.4.57 void Parma_Polyhedra_Library::gcdext_assign (GMP_Integer & *x*, GMP_Integer & *s*, GMP_Integer & *t*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 62 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::reduce_pc_with_pc().

10.1.4.58 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::greater_or_equal (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.59 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::greater_than (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.60 void Parma_Polyhedra_Library::initialize () [inline]

Initializes the library.

Definition at line 47 of file initializer.hh.

Referenced by Parma_Polyhedra_Library::Init::Init().

10.1.4.61 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 579 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder, Parma_Polyhedra_Library::Checked::Result, and set_integer().

10.1.4.62 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 256 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.63 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 86 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign().

10.1.4.64 template<typename ITV > Poly_Con_Relation Parma_Polyhedra_Library::interval_relation (const ITV & *i*, const Constraint::Type *constraint_type*, Coefficient_traits::const_reference *num*, Coefficient_traits::const_reference *den* = 1) [inline]

Returns the relations holding between an interval and an interval constraint.

Parameters

i The interval;

constraint_type The constraint type;

num The numerator of the constraint bound;

den The denominator of the constraint bound

The interval constraint has the form `den * Variable(0) relsym num` where relsym is ==, > or >= depending on the `constraint_type`.

Definition at line 584 of file Box.templates.hh.

References assign_r(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, and Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects().

Referenced by Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), and Parma_Polyhedra_Library::Box< ITV >::relation_with().

10.1.4.65 unsigned Parma_Polyhedra_Library::irrational_precision () [inline]

Returns the precision parameter used for irrational calculations.

Definition at line 534 of file checked_mpq.inlines.hh.

References Parma_Polyhedra_Library::Checked::irrational_precision.

10.1.4.66 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_additive_inverse (const T & *x*, const T & *y*) [inline]

Returns `true` if and only if $x = -y$.

Definition at line 102 of file math_utilities.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::compute_predecessors(), Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_successors(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::operator<<(),

Parma_Polyhedra_Library::BD_Shape< T >::operator<<(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_-Library::Octagonal_Shape< T >::tight_coherence_would_make_empty().

10.1.4.67 bool Parma_Polyhedra_Library::is_canonical (const mpq_class & *x*) [inline]

Returns `true` if and only if `x` is in canonical form.

Definition at line 109 of file math_utilities.inlines.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_-Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Box< ITV >::max_min(), and Parma_Polyhedra_Library::MIP_Problem::solve_mip().

10.1.4.68 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_even (const T & *x*) [inline]

Returns `true` if and only if `x` is an even number.

Definition at line 94 of file math_utilities.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), and Parma_Polyhedra_-Library::Octagonal_Shape< T >::tight_coherence_would_make_empty().

10.1.4.69 template<typename Boundary , typename Info > int Parma_-Polyhedra_Library::is_infinity (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 73 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_infinity().

10.1.4.70 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, int>::type Parma_Polyhedra_Library::is_infinity (const T & *x*) [related]

Definition at line 239 of file Checked_Number.inlines.hh.

References is_minus_infinity(), and is_plus_infinity().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.71 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, bool>::type Parma_Polyhedra_Library::is_integer (const T & *x*) [related]

Definition at line 252 of file Checked_Number.inlines.hh.

References raw_value().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::get(), Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::get(), Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::get(), Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::get(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::integer_upper_bound_assign_if_exact(), is_singleton_integer(), Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::restrict(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty().

10.1.4.72 template<typename T , typename Policy > bool Parma_Polyhedra_Library::is_minus_infinity (const Checked_Number< T, Policy > & *x*) [inline]

Definition at line 310 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

10.1.4.73 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, bool>::type Parma_Polyhedra_Library::is_minus_infinity (const T & *x*) [related]

Definition at line 225 of file Checked_Number.inlines.hh.

References raw_value().

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::ascii_load(), Parma_Polyhedra_Library::DB_Matrix< T >::ascii_load(), is_infinity(), numer_denom(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::BD_Shape< T >::OK().

10.1.4.74 template<typename T , typename Policy > bool Parma_Polyhedra_Library::is_not_a_number (const Checked_Number< T, Policy > & *x*) [inline]

Definition at line 304 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

10.1.4.75 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, bool>::type Parma_Polyhedra_Library::is_not_a_number (const T & *x*) [related]

Definition at line 245 of file Checked_Number.inlines.hh.

References raw_value().

Referenced by Parma_Polyhedra_Library::Interval_NS::f_is_empty(), numer_denom(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::DB_Row< T >::OK().

10.1.4.76 template<typename T , typename Policy > bool Parma_Polyhedra_Library::is_plus_infinity (const Checked_Number< T, Policy > & *x*) [inline]

Definition at line 316 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

10.1.4.77 template<typename T > Enable_If<Is_Native_Or_Checked<T>::value, bool>::type Parma_Polyhedra_Library::is_plus_infinity (const T & *x*) [related]

Definition at line 232 of file Checked_Number.inlines.hh.

References raw_value().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape<

T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_-integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_-path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_-closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_-Library::BD_Shape< T >::is_bounded(), is_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::is_-shortest_path_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::BD_Shape< T >::is_-universe(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_-Library::DB_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_-Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), numer_denom(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::operator<<(), Parma_Polyhedra_-Library::BD_Shape< T >::operator<<(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_-Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_-using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::tight_closure_assign(), and Parma_Polyhedra_Library::Octagonal_-Shape< T >::tight_coherence_would_make_empty().

10.1.4.78 template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, bool>::type Parma_Polyhedra_Library::is_singleton_integer (const T & *x*) [inline]

Definition at line 144 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval_NS::f_lower(), and is_integer().

10.1.4.79 unsigned long Parma_Polyhedra_Library::isqrt (unsigned long *x*) [inline]

Returns the integer square root of x.

Definition at line 409 of file OR_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::max_num_rows().

10.1.4.80 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 528 of file Interval_Restriction.defs.hh.

References assign_r(), contains_restriction(), Parma_Polyhedra_Library::Interval_Restriction_-

Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder, Parma_Polyhedra_Library::Checked::Result, set_integer(), and set_unrestricted().

10.1.4.81 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 248 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.82 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 80 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign().

10.1.4.83 void Parma_Polyhedra_Library::lcm_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 70 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Grid::normalize_divisors(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), and Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index().

10.1.4.84 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::less_or_equal (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.85 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::less_than (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.86 template<typename T > T Parma_Polyhedra_Library::low_bits_mask (unsigned *n*) [inline]

Returns a mask for the lowest n bits,.

Definition at line 45 of file math_utilities.inlines.hh.

10.1.4.87 template<typename N > void Parma_Polyhedra_Library::max_assign (N & *x*, const N & *y*) [inline]

Assigns to x the maximum between x and y.

Definition at line 87 of file math_utilities.inlines.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

10.1.4.88 void Parma_Polyhedra_Library::maybe_abandon () [related]

Definition at line 57 of file globals.inlines.hh.

References abandon_expensive_computations, Parma_Polyhedra_Library::Weightwatch_Traits::check_function, and Parma_Polyhedra_Library::Implementation::in_assert.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_exact_pricing(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), and Parma_Polyhedra_Library::PIP_Solution_Node::solve().

10.1.4.89 template<typename To , typename From > Result Parma_Polyhedra_Library::maybe_assign (const To *& *top*, To & *tmp*, const From & *from*, Rounding_Dir *dir*) [inline]

Assigns to top a pointer to a location that holds the conversion, according to dir, of from to type To. When necessary, and only when necessary, the variable tmp is used to hold the result of conversion.

Definition at line 64 of file distances.inlines.hh.

References Parma_Polyhedra_Library::maybe_assign_struct< To, From >::function().

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Generator::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_distance_assign(), and Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign().

10.1.4.90 template<typename T > int Parma_Polyhedra_Library::maybe_check_fpu_inexact () [inline]

Definition at line 835 of file Checked_Number.inlines.hh.

10.1.4.91 template<typename T > void Parma_Polyhedra_Library::maybe_reset_fpu_inexact () [inline]

Definition at line 828 of file Checked_Number.inlines.hh.

10.1.4.92 template<typename N > void Parma_Polyhedra_Library::min_assign (N & *x*, const N & *y*) [inline]

Assigns to x the minimum between x and y.

Definition at line 80 of file math_utilities.inlines.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign().

10.1.4.93 template<typename T > T Parma_Polyhedra_Library::minus_infinity () [inline]

Definition at line 808 of file Checked_Number.inlines.hh.

References MINUS_INFINITY.

10.1.4.94 void Parma_Polyhedra_Library::mul_2exp_assign (GMP_Integer & *x*, const GMP_Integer & *y*, unsigned int *exp*) [related]

Definition at line 85 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

10.1.4.95 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 800 of file Interval_Restriction.defs.hh.

References assign_r(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_NS::f_lower(), mulmod(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder, Parma_Polyhedra_Library::Checked::Result, set_integer(), and set_unrestricted().

10.1.4.96 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 295 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.97 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 116 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign().

10.1.4.98 template<typename T > void Parma_Polyhedra_Library::mulmod (T & *to*, const T & *x*, const T & *y*, const T & *to_m*) [inline]

Definition at line 771 of file Interval_Restriction.defs.hh.

References assign_r(), and Parma_Polyhedra_Library::Checked::Result.

Referenced by mul_restriction().

10.1.4.99 template<typename From > Enable_If< Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::neg_assign (From & *x*) [inline]

Definition at line 31 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Checked::From.

10.1.4.100 void Parma_Polyhedra_Library::neg_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Definition at line 37 of file GMP_Integer.inlines.hh.

10.1.4.101 void Parma_Polyhedra_Library::neg_assign (GMP_Integer & *x*) [related]

Definition at line 32 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Polyhedron::affine_image(), Parma_Polyhedra_Library::Grid::affine_image(), Parma_Polyhedra_Library::BD_Shape<

T >::affine_image(), Parma_Polyhedra_Library::Polyhedron::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Grid::affine_preimage(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::Linear_System::back_substitute(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Grid_Generator::negate(), Parma_Polyhedra_Library::Congruence::negate(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Linear_Expression::operator<<(), Parma_Polyhedra_Library::Grid_Generator::operator<<(), Parma_Polyhedra_Library::Generator::operator<<(), Parma_Polyhedra_Library::Constraint::operator<<(), Parma_Polyhedra_Library::Congruence::operator<<(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), Parma_Polyhedra_Library::Linear_Row::sign_normalize(), Parma_Polyhedra_Library::Congruence::sign_normalize(), Parma_Polyhedra_Library::PIP_Problem::solve(), Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

10.1.4.102 template<typename T , typename Base , typename From > bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From & *x*) [inline]

Definition at line 626 of file Interval_Restriction.defs.hh.

References assign_restriction().

10.1.4.103 template<typename Base , typename From > bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_Integer< Base > & *to*, const From & *x*) [inline]

Definition at line 272 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.104 template<typename T > bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_None_Base &, const T &) [inline]

Definition at line 98 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign().

10.1.4.105 void Parma_Polyhedra_Library::normalize2 (Coefficient_traits::const_reference *x*, Coefficient_traits::const_reference *y*, Coefficient & *nx*, Coefficient & *ny*) [inline]

If g is the GCD of `x` and `y`, the values of `x` and `y` divided by g are assigned to `nx` and `ny`, respectively.

Note

`x` and `nx` may be the same object and likewise for `y` and `ny`. Any other aliasing results in undefined behavior.

Definition at line 34 of file math_utilities.inlines.hh.

References exact_div_assign(), gcd_assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), and Parma_Polyhedra_Library::Linear_Row::linear_combine().

10.1.4.106 dimension_type Parma_Polyhedra_Library::not_a_dimension () [inline]

Returns a value that does not designate a valid dimension.

Definition at line 33 of file globals.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Problem::clear(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), Parma_Polyhedra_Library::Variable::max_space_dimension(), Parma_Polyhedra_Library::MIP_Problem::merge_split_variable(), Parma_Polyhedra_Library::PIP_Problem::OK(), Parma_Polyhedra_Library::PIP_Problem::operator<<(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::PIP_Solution_Node::row_sign(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

10.1.4.107 template<typename T > T Parma_Polyhedra_Library::not_a_number () [inline]

Definition at line 814 of file Checked_Number.inlines.hh.

References NOT_A_NUMBER.

10.1.4.108 template<typename T1 , typename T2 > Enable_If<Is_Native_Or_Checked<T1>::value && Is_Native_Or_Checked<T2>::value, bool>::type Parma_Polyhedra_Library::not_equal (const T1 & *x*, const T2 & *y*) [inline]

10.1.4.109 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::numer_denom (const T & *from*, Coefficient & *num*, Coefficient & *den*) [inline]

Extract the numerator and denominator components of `from`.

Definition at line 52 of file math_utilities.inlines.hh.

References assign_r(), is_minus_infinity(), is_not_a_number(), and is_plus_infinity().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

10.1.4.110 template<typename PSET > bool Parma_Polyhedra_Library::one_affine_ranking_function_MS (const PSET & *pset*, Generator & *mu*) [inline]

Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset A pointset approximating the behavior of a loop whose termination is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

where unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update.

mu When `true` is returned, this is assigned a point of space dimension $n + 1$ encoding one (not further specified) affine ranking function for the loop being analyzed. The ranking function is of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ are the coefficients of `mu` corresponding to the space dimensions $n, 0, \ldots, n-1$, respectively.

Returns

`true` if any loop approximated by `pset` definitely terminates; `false` if the test is inconclusive. However, if `pset` *precisely* characterizes the effect of the loop body onto the loop-relevant program variables, then `true` is returned *if and only if* the loop terminates.

Definition at line 259 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation().

Referenced by one_affine_ranking_function_MS_2().

10.1.4.111 template<typename PSET > bool Parma_Polyhedra_Library::one_affine_ranking_function_MS_2 (const PSET & *pset_before*, const PSET & *pset_after*, Generator & *mu*) [inline]

Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset_before A pointset approximating the values of loop-relevant variables *before* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x_1, \ldots, x_n$ go onto space dimensions $0, \ldots, n-1$.

pset_after A pointset approximating the values of loop-relevant variables *after* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

Note that unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update. Note also that unprimed variables are assigned to different space dimensions in `pset_before` and `pset_after`.

Parameters

mu When `true` is returned, this is assigned a point of space dimension $n + 1$ encoding one (not further specified) affine ranking function for the loop being analyzed. The ranking function is of the form $\mu_0 + \sum_{i=1}^{n} \mu_i x_i$ where $\mu_0, \mu_1, \ldots, \mu_n$ are the coefficients of `mu` corresponding to the space dimensions $n, 0, \ldots, n-1$, respectively.

Returns

`true` if any loop approximated by `pset` definitely terminates; `false` if the test is inconclusive. However, if `pset_before` and `pset_after` *precisely* characterize the effect of the loop body onto the loop-relevant program variables, then `true` is returned *if and only if* the loop terminates.

Definition at line 277 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and one_affine_ranking_function_MS().

10.1.4.112 template<typename PSET > bool Parma_Polyhedra_Library::one_affine_ranking_function_PR (const PSET & *pset*, Generator & *mu*) [inline]

Like one_affine_ranking_function_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 476 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR_original().

Referenced by one_affine_ranking_function_PR_2().

10.1.4.113 template<typename PSET > bool Parma_Polyhedra_Library::one_affine_ranking_function_PR_2 (const PSET & *pset_before*, const PSET & *pset_after*, Generator & *mu*) [inline]

Like one_affine_ranking_function_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 451 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and one_affine_ranking_function_PR().

10.1.4.114 template<typename ITV > bool Parma_Polyhedra_Library::operator!= (const Box< ITV > & *x*, const Box< ITV > & *y*) [related]

Definition at line 254 of file Box.inlines.hh.

10.1.4.115 template<typename T1 , typename T2 > Enable_If<((Is_Singleton<T1>::value || Is_Interval<T1>::value) && (Is_Singleton<T2>::value || Is_Interval<T2>::value) && (Is_Interval<T1>::value || Is_Interval<T2>::value)), bool>::type Parma_Polyhedra_Library::operator!= (const T1 & *x*, const T2 & *y*) [related]

Definition at line 183 of file Interval.inlines.hh.

10.1.4.116 template<typename B , typename Info > Interval<B, Info> Parma_Polyhedra_Library::operator∗ (const Interval< B, Info > & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1035 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign().

10.1.4.117 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator∗ (const T & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1027 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign().

10.1.4.118 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator∗ (const Interval< B, Info > & *x*, const T & *y*) [inline]

Definition at line 1019 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign().

10.1.4.119 template<typename B , typename Info > Interval<B, Info> Parma_Polyhedra_Library::operator+ (const Interval< B, Info > & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 987 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign().

10.1.4.120 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator+ (const T & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 979 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign().

10.1.4.121 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator+ (const Interval< B, Info > & *x*, const T & *y*) [inline]

Definition at line 971 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign().

10.1.4.122 template<typename B , typename Info > Interval<B, Info> Parma_Polyhedra_Library::operator- (const Interval< B, Info > & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1011 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.123 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator- (const T & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1003 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.124 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator- (const Interval< B, Info > & *x*, const T & *y*) [inline]

Definition at line 995 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.125 template<typename B , typename Info > Interval<B, Info> Parma_Polyhedra_Library::operator/ (const Interval< B, Info > & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1059 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign().

10.1.4.126 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator/ (const T & *x*, const Interval< B, Info > & *y*) [inline]

Definition at line 1051 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign().

10.1.4.127 template<typename B , typename Info , typename T > Enable_If<Is_Singleton<T>::value, Interval<B, Info> >::type Parma_Polyhedra_Library::operator/ (const Interval< B, Info > & *x*, const T & *y*) [inline]

Definition at line 1043 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign().

10.1.4.128 template<typename Boundary , typename Info > std::ostream& Parma_Polyhedra_Library::operator<< (std::ostream & *os*, const Interval< Boundary, Info > & *x*) [inline]

Definition at line 1067 of file Interval.inlines.hh.

References check_empty_arg(), and output_restriction().

10.1.4.129 template<typename ITV > bool Parma_Polyhedra_Library::operator== (const Box< ITV > & *x*, const Box< ITV > & *y*) [inline]

Definition at line 533 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

10.1.4.130 template<typename T1 , typename T2 > Enable_If<((Is_Singleton<T1>::value || Is_Interval<T1>::value) && (Is_Singleton<T2>::value || Is_Interval<T2>::value) && (Is_Interval<T1>::value || Is_Interval<T2>::value)), bool>::type Parma_Polyhedra_Library::operator== (const T1 & *x*, const T2 & *y*) [related]

Definition at line 164 of file Interval.inlines.hh.

References check_empty_arg(), Parma_Polyhedra_Library::Checked::eq, eq_restriction(), Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), f_OK(), and Parma_Polyhedra_Library::Interval_NS::f_upper().

Referenced by Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator!=().

10.1.4.131 template<typename T , typename Base > void Parma_Polyhedra_Library::output_restriction (std::ostream & *s*, const Interval_Restriction_Integer_Modulo< T, Base > & *x*) [inline]

Definition at line 865 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

10.1.4.132 template<typename Base > void Parma_Polyhedra_Library::output_restriction (std::ostream & *s*, const Interval_Restriction_Integer< Base > & *x*) [inline]

Definition at line 310 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::get_integer().

10.1.4.133 void Parma_Polyhedra_Library::output_restriction (std::ostream &, const Interval_Restriction_None_Base &) [inline]

Definition at line 127 of file Interval_Restriction.defs.hh.

Referenced by operator<<().

10.1.4.134 template<typename T > T Parma_Polyhedra_Library::plus_infinity () [inline]

Definition at line 802 of file Checked_Number.inlines.hh.

References PLUS_INFINITY.

10.1.4.135 mpz_class& Parma_Polyhedra_Library::raw_value (GMP_Integer & *x*) [related]

Definition at line 116 of file GMP_Integer.inlines.hh.

10.1.4.136 const mpz_class& Parma_Polyhedra_Library::raw_value (const GMP_Integer & *x*) [related]

Definition at line 111 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::assign_r(), assign_r(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::cmp(), construct(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::input(), is_integer(), is_minus_infinity(), is_not_a_number(), is_plus_infinity(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::output(), Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >::raw_value(), Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value

>::type >::raw_value(), Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >::raw_value(), Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >::raw_value(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::sgn().

10.1.4.137 void Parma_Polyhedra_Library::rem_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 57 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux().

10.1.4.138 void Parma_Polyhedra_Library::restore_pre_PPL_rounding () [inline]

Sets the FPU rounding mode as it was before initialization of the PPL.

After calling this function it is absolutely necessary to call set_rounding_for_PPL() before using any PPL abstractions based on floating point numbers. This is performed automatically at finalization-time.

Definition at line 40 of file Init.inlines.hh.

References Parma_Polyhedra_Library::Init::old_rounding_direction.

10.1.4.139 Rounding_Dir Parma_Polyhedra_Library::rounding_dir (Rounding_Dir *dir*) [inline]

Definition at line 38 of file Checked_Number.inlines.hh.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::assign_r(), assign_r(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number(), construct(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::input(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::output().

10.1.4.140 template<typename T , typename Base > bool Parma_Polyhedra_Library::set_integer (Interval_Restriction_Integer_Modulo< T, Base > & *to*) [inline]

Definition at line 512 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

Referenced by add_restriction(), intersect_restriction(), join_restriction(), mul_restriction(), and sub_restriction().

10.1.4.141 void Parma_Polyhedra_Library::set_irrational_precision (const unsigned *p*) [inline]

Sets the precision parameter used for irrational calculations.

The lesser between numerator and denominator is limited to 2**p.

If p is less than or equal to INT_MAX, sets the precision parameter used for irrational calculations to p.

Exceptions

std::invalid_argument Thrown if p is greater than INT_MAX.

Definition at line 548 of file checked_mpq.inlines.hh.

References Parma_Polyhedra_Library::Checked::irrational_precision.

Referenced by Parma_Polyhedra_Library::Init::Init().

10.1.4.142 void Parma_Polyhedra_Library::set_rounding_for_PPL () [inline]

Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

This is performed automatically at initialization-time. Calling this function is needed only if restore_pre_PPL_rounding() has been previously called.

Definition at line 33 of file Init.inlines.hh.

10.1.4.143 template<typename T , typename Base > bool Parma_Polyhedra_Library::set_unrestricted (Interval_Restriction_Integer_Modulo< T, Base > & *to*) [inline]

Definition at line 504 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

Referenced by add_restriction(), div_restriction(), join_restriction(), mul_restriction(), and sub_restriction().

10.1.4.144 template<typename D1 , typename D2 > bool Parma_Polyhedra_Library::shrink_to_congruence_no_check (D1 & *d1*, D2 & *d2*, const Congruence & *cg*) [inline]

Definition at line 497 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Congruence::coefficient(), EMPTY, Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Congruence::space_dimension(), and swap().

Referenced by Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >::product_reduce().

10.1.4.145 void Parma_Polyhedra_Library::sqrt_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Definition at line 101 of file GMP_Integer.inlines.hh.

10.1.4.146 void Parma_Polyhedra_Library::sub_mul_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Definition at line 80 of file GMP_Integer.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), Parma_Polyhedra_Library::Linear_Row::linear_combine(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Grid::reduce_equality_with_equality(), Parma_Polyhedra_Library::Grid::reduce_line_with_line(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_Library::Grid::reduce_pc_with_pc(), and Parma_Polyhedra_Library::Grid::reduce_reduced().

10.1.4.147 template<typename T , typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_Integer_Modulo< T, Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 734 of file Interval_Restriction.defs.hh.

References assign_r(), assign_rem(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder, Parma_Polyhedra_Library::Checked::Result, set_integer(), set_unrestricted(), and submod().

10.1.4.148 template<typename Base , typename From1 , typename From2 > bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_Integer< Base > & *to*, const From1 & *x*, const From2 & *y*) [inline]

Definition at line 287 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer().

10.1.4.149 template<typename T1 , typename T2 > bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &) [inline]

Definition at line 110 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.4.150 template<typename T > void Parma_Polyhedra_Library::submod (T & *to*, const T & *x*, const T & *y*, const T & *to_m*, const T & *y_m*) [inline]

Definition at line 716 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Checked::Result.

Referenced by sub_restriction().

10.1.4.151 template<typename T > Enable_If<Slow_Copy<T>::value, void>::type Parma_Polyhedra_Library::swap (T &, T &) [inline]

Make sure swap() is specialized when needed.

This will cause a compile-time error whenever a specialization for `T` is beneficial but missing.

Definition at line 51 of file globals.defs.hh.

References PPL_COMPILE_TIME_CHECK.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Grid_Generator::coefficient_swap(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::Congruence_System::concatenate(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Row::iter_swap(), Parma_Polyhedra_Library::Linear_Row::iter_swap(), Parma_Polyhedra_Library::DB_Row< T >::iter_swap(), Parma_Polyhedra_Library::Bit_Row::iter_swap(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product(), Parma_Polyhedra_Library::Matrix::permute_columns(), Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), shrink_to_congruence_no_check(), Parma_Polyhedra_Library::Linear_System::simplify(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), Parma_Polyhedra_Library::PIP_Problem::swap(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::swap(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign_if_exact().

10.1.4.152 template<typename PSET > bool Parma_Polyhedra_Library::termination_test_MS (const PSET & *pset*) [inline]

Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset A pointset approximating the behavior of a loop whose termination is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

where unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update.

Returns

`true` if any loop approximated by `pset` definitely terminates; `false` if the test is inconclusive. However, if `pset` *precisely* characterizes the effect of the loop body onto the loop-relevant program variables, then `true` is returned *if and only if* the loop terminates.

Definition at line 221 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation().

Referenced by termination_test_MS_2().

10.1.4.153 template<typename PSET > bool Parma_Polyhedra_Library::termination_test_MS_2 (const PSET & *pset_before*, const PSET & *pset_after*) `[inline]`

Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

Template Parameters

PSET Any pointset supported by the PPL that provides the `minimized_constraints()` method.

Parameters

pset_before A pointset approximating the values of loop-relevant variables *before* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x_1, \ldots, x_n$ go onto space dimensions $0, \ldots, n-1$.

pset_after A pointset approximating the values of loop-relevant variables *after* the update performed in the loop body that is being analyzed. The variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$,

Note that unprimed variables represent the values of the loop-relevant program variables before the update performed in the loop body, and primed variables represent the values of those program variables after the update. Note also that unprimed variables are assigned to different space dimensions in `pset_before` and `pset_after`.

Returns

`true` if any loop approximated by `pset` definitely terminates; `false` if the test is inconclusive. However, if `pset_before` and `pset_after` *precisely* characterize the effect of the loop body onto the loop-relevant program variables, then `true` is returned *if and only if* the loop terminates.

Definition at line 239 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and termination_test_MS().

10.1.4.154 template<typename PSET > bool Parma_Polyhedra_Library::termination_test_PR (const PSET & *pset*) [inline]

Like termination_test_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 433 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR_original().

Referenced by termination_test_PR_2().

10.1.4.155 template<typename PSET > bool Parma_Polyhedra_Library::termination_test_PR_2 (const PSET & *pset_before*, const PSET & *pset_after*) [inline]

Like termination_test_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

Definition at line 411 of file termination.templates.hh.

References Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), and termination_test_PR().

10.1.4.156 void Parma_Polyhedra_Library::throw_result_exception (Result *r*)

Definition at line 30 of file Checked_Number.cc.

Referenced by Parma_Polyhedra_Library::Extended_Number_Policy::handle_result().

10.1.4.157 memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpq_class & *x*) [inline]

Returns the total size in bytes of the memory occupied by `x`.

Definition at line 113 of file globals.inlines.hh.

References external_memory_in_bytes().

10.1.4.158 memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpz_class & *x*) [inline]

Returns the total size in bytes of the memory occupied by x.

Definition at line 102 of file globals.inlines.hh.

References external_memory_in_bytes().

10.1.4.159 template<typename T > Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_Library::total_memory_in_bytes (const T &) [inline]

For native types, returns the total size in bytes of the memory occupied by the type of the (unused) parameter, i.e., 0.

Definition at line 92 of file globals.inlines.hh.

10.1.4.160 memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const GMP_Integer & *x*)

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::total_memory_in_bytes().

10.1.4.161 const char * Parma_Polyhedra_Library::version ()

Returns a character string containing the PPL version.

Definition at line 99 of file version.cc.

10.1.4.162 unsigned Parma_Polyhedra_Library::version_beta ()

Returns the beta number of the PPL version.

Definition at line 94 of file version.cc.

References PPL_VERSION_BETA.

10.1.4.163 unsigned Parma_Polyhedra_Library::version_major ()

Returns the major number of the PPL version.

Definition at line 79 of file version.cc.

References PPL_VERSION_MAJOR.

10.1.4.164 unsigned Parma_Polyhedra_Library::version_minor ()

Returns the minor number of the PPL version.

Definition at line 84 of file version.cc.

References PPL_VERSION_MINOR.

10.1.4.165 unsigned Parma_Polyhedra_Library::version_revision ()

Returns the revision number of the PPL version.

Definition at line 89 of file version.cc.

References PPL_VERSION_REVISION.

10.1.5 Variable Documentation

10.1.5.1 Minus_Infinity Parma_Polyhedra_Library::MINUS_INFINITY

Definition at line 30 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), minus_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.1.5.2 Not_A_Number Parma_Polyhedra_Library::NOT_A_NUMBER

Definition at line 32 of file checked.cc.

Referenced by not_a_number().

10.1.5.3 Plus_Infinity Parma_Polyhedra_Library::PLUS_INFINITY

Definition at line 31 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_binary_octagonal_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(),

Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.2 Parma_Polyhedra_Library::Checked Namespace Reference

Types and functions implementing checked numbers.

Classes

- struct Safe_Conversion
- struct Safe_Conversion< T, T >
- struct Safe_Int_Comparison
- struct Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >
- struct Float_2exp
- struct Extended_Int
- struct Larger< char >
- struct Larger< signed char >
- struct Larger< unsigned char >
- struct Larger< signed short >
- struct Larger< unsigned short >
- struct Larger< signed int >
- struct Larger< unsigned int >
- struct Larger< signed long >
- struct Larger< unsigned long >
- struct Larger< signed long long >
- struct Larger< unsigned long long >
- struct number_struct

Typedefs

- typedef int16_t int_fast16_t
- typedef int32_t int_fast32_t
- typedef int64_t int_fast64_t
- typedef uint16_t uint_fast16_t
- typedef uint32_t uint_fast32_t
- typedef uint64_t uint_fast64_t
- typedef int mp_size_field_t

  This is assumed to be the type of the _mp_size field of GMP's __mpz_struct.

Functions

- PPL_DECLARE_FUN2_0_0 (copy, void, nonconst, Type1, const, Type2) PPL_DECLARE_-FUN1_0_0(sgn
- From PPL_DECLARE_FUN2_0_0 (cmp, Result_Relation, const, Type1, const, Type2) PPL_-DECLARE_FUN1_0_3(classify
- From bool PPL_DECLARE_FUN1_0_0 (is_nan, bool, const, Type) PPL_DECLARE_FUN1_0_-0(is_minf
- From bool Type PPL_DECLARE_FUN1_0_0 (is_pinf, bool, const, Type) PPL_DECLARE_-FUN1_0_0(is_int
- From bool Type Type PPL_DECLARE_FUN1_0_2 (assign_special, Result, nonconst, Type, Result_Class, Rounding_Dir) PPL_DECLARE_FUN1_0_2(construct_special
- From bool Type Type Rounding_Dir PPL_DECLARE_FUN2_0_1 (construct, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_0_1(assign
- From bool Type Type Rounding_Dir Rounding_Dir PPL_DECLARE_FUN2_0_1 (floor, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_0_1(ceil
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir PPL_DECLARE_FUN2_0_1 (trunc, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_0_1(neg
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir PPL_-DECLARE_FUN2_0_1 (abs, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_-FUN2_0_1(sqrt
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir PPL_DECLARE_FUN3_0_1 (add, Result, nonconst, To, const, From1, const, From2, Rounding_-Dir) PPL_DECLARE_FUN3_0_1(sub
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_-Dir Rounding_Dir PPL_DECLARE_FUN3_0_1 (mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(div
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_-Dir Rounding_Dir Rounding_Dir PPL_DECLARE_FUN3_0_1 (rem, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(idiv
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir PPL_DECLARE_FUN2_0_2 (add_2exp, Result, non-const, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_FUN2_0_2(sub_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir PPL_DECLARE_FUN2_0_-2 (mul_2exp, Result, nonconst, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_-FUN2_0_2(div_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_-Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_-Dir PPL_DECLARE_FUN2_0_2 (smod_2exp, Result, nonconst, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_FUN2_0_2(umod_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir un-signed Rounding_Dir PPL_DECLARE_FUN3_0_1 (add_mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(sub_mul
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir un-signed Rounding_Dir Rounding_Dir PPL_DECLARE_FUN3_0_1 (gcd, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN5_0_1(gcdext
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir un-signed Rounding_Dir Rounding_Dir Rounding_Dir PPL_DECLARE_FUN3_0_1 (lcm, Result, non-const, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN1_0_2(input

- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir PPL_DECLARE_-FUN1_1_2 (output, Result, std::ostream &, const, Type, const Numeric_Format &, Rounding_Dir) template< typename Policy
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir To Result round (To &to, Result r, Rounding_Dir dir)
- Result input_mpq (mpq_class &to, std::istream &is)
- PPL_SAFE_CONVERSION (signed short, signed char)
- PPL_SAFE_CONVERSION (signed long long, signed char)
- PPL_SAFE_CONVERSION (signed long long, signed short)
- PPL_SAFE_CONVERSION (unsigned short, char)
- PPL_SAFE_CONVERSION (unsigned short, unsigned char)
- PPL_SAFE_CONVERSION (unsigned long, char)
- PPL_SAFE_CONVERSION (unsigned long, unsigned char)
- PPL_SAFE_CONVERSION (unsigned long long, char)
- PPL_SAFE_CONVERSION (unsigned long long, unsigned char)
- PPL_SAFE_CONVERSION (double, float)
- PPL_SAFE_CONVERSION (long double, float)
- PPL_SAFE_CONVERSION (long double, double)
- PPL_SAFE_CONVERSION (mpz_class, char)
- PPL_SAFE_CONVERSION (mpz_class, signed char)
- PPL_SAFE_CONVERSION (mpz_class, unsigned char)
- PPL_SAFE_CONVERSION (mpq_class, char)
- PPL_SAFE_CONVERSION (mpq_class, signed char)
- PPL_SAFE_CONVERSION (mpq_class, unsigned char)
- PPL_SAFE_CONVERSION (mpq_class, float)
- PPL_SAFE_CONVERSION (mpq_class, double)
- template<typename Policy , typename Type >
  struct PPL_FUNCTION_CLASS (construct)< Policy
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  struct PPL_FUNCTION_CLASS (construct)
- template<typename To_Policy , typename To >
  struct PPL_FUNCTION_CLASS (construct_special)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_exact (To &to, const From &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Enable_If< Is_Same< To_Policy, From_Policy >::value, void >::type copy_generic (Type &to, const Type &from)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result abs_generic (To &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From >
  void gcd_exact_noabs (To &to, const From &x, const From &y)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 >
  Result gcd_exact (To &to, const From1 &x, const From2 &y, Rounding_Dir dir)
- template<typename To1_Policy , typename To2_Policy , typename To3_Policy , typename From1_Policy , typename From2_Policy , typename To1 , typename To2 , typename To3 , typename From1 , typename From2 >
  Result gcdext_exact (To1 &to, To2 &s, To3 &t, const From1 &x, const From2 &y, Rounding_Dir dir)

- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 >
  Result lcm_gcd_exact (To &to, const From1 &x, const From2 &y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result_Relation sgn_generic (const Type &x)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type lt (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type le (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type eq (const T1 &x, const T2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool lt_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool le_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool eq_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  Result_Relation cmp_generic (const Type1 &x, const Type2 &y)
- template<typename Policy , typename Type >
  Result assign_nan (Type &to, Result r)
- template<typename Policy , typename Type >
  Result input_generic (Type &to, std::istream &is, Rounding_Dir dir)
- float fma (float x, float y, float z)
- double fma (double x, double y, double z)
- long double fma (long double x, long double y, long double z)
- double rint (double x)
- long double rint (long double x)
- bool fpu_direct_rounding (Rounding_Dir dir)
- bool fpu_inverse_rounding (Rounding_Dir dir)
- void limit_precision (const float &v)
- void limit_precision (const double &v)
- void limit_precision (const long double &)
- template<typename Policy , typename T >
  Result classify_float (const T v, bool nan, bool inf, bool sign)
- template<typename Policy , typename T >
  bool is_nan_float (const T v)
- template<typename Policy , typename T >
  int is_inf_float (const T v)
- template<typename Policy , typename T >
  bool is_minf_float (const T v)
- template<typename Policy , typename T >
  bool is_pinf_float (const T v)
- template<typename Policy , typename T >
  bool is_int_float (const T v)
- template<typename Policy , typename T >
  Result assign_special_float (T &v, Result_Class c, Rounding_Dir)
- template<typename T >
  void pred_float (T &v)

- template<typename T >
  void succ_float (T &v)
- template<typename Policy , typename To >
  Result round_lt_float (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result round_gt_float (To &to, Rounding_Dir dir)
- template<typename Policy >
  void prepare_inexact (Rounding_Dir dir)
- template<typename Policy >
  Result result_relation (Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_float_float_exact (To &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_float_float_inexact (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_float_float (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result floor_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result ceil_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result trunc_float (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result neg_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result div_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result idiv_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result rem_float (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result add_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sub_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result mul_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result div_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result smod_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result umod_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result abs_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sqrt_float (Type &to, const Type from, Rounding_Dir dir)

- template<typename Policy , typename Type >
  Result_Relation sgn_float (const Type x)
- template<typename Policy1 , typename Policy2 , typename Type >
  Result_Relation cmp_float (const Type x, const Type y)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_float_int_inexact (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_float_int (To &to, const From from, Rounding_Dir dir)
- template<typename Policy , typename T >
  Result set_neg_overflow_float (T &to, Rounding_Dir dir)
- template<typename Policy , typename T >
  Result set_pos_overflow_float (T &to, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename T >
  Result assign_float_mpz (T &to, const mpz_class &_from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename T >
  Result assign_float_mpq (T &to, const mpq_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result output_float (std::ostream &os, const Type from, const Numeric_Format &, Rounding_Dir)
- template<typename Policy , typename To >
  Result set_neg_overflow_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result set_pos_overflow_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result round_lt_int_no_overflow (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result round_gt_int_no_overflow (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result round_lt_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result round_gt_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result classify_int (const Type v, bool nan, bool inf, bool sign)
- template<typename Policy , typename Type >
  bool is_nan_int (const Type v)
- template<typename Policy , typename Type >
  bool is_minf_int (const Type v)
- template<typename Policy , typename Type >
  bool is_pinf_int (const Type v)
- template<typename Policy , typename Type >
  bool is_int_int (const Type v)
- template<typename Policy , typename Type >
  Result assign_special_int (Type &v, Result_Class c, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_signed_int_signed_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_signed_int_unsigned_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_unsigned_int_signed_int (To &to, const From from, Rounding_Dir dir)

- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_unsigned_int_unsigned_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result assign_int_float (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result assign_signed_int_mpz (To &to, const mpz_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result assign_unsigned_int_mpz (To &to, const mpz_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result assign_int_mpq (To &to, const mpq_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result neg_int_larger (Type &to, const Type x, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result mul_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result neg_signed_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result neg_unsigned_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result mul_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result mul_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result div_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result div_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result idiv_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result idiv_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result rem_signed_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result rem_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result div_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result div_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result add_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)

- template<typename To_Policy , typename From_Policy , typename Type >
  Result add_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sub_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sub_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result mul_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result mul_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result smod_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result smod_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result umod_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result umod_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename Type >
  void isqrtrem (Type &q, Type &r, const Type from)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sqrt_unsigned_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result sqrt_signed_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result add_mul_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result sub_mul_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result output_char (std::ostream &os, Type &from, const Numeric_Format &, Rounding_Dir)
- template<typename Policy , typename Type >
  Result output_int (std::ostream &os, Type &from, const Numeric_Format &, Rounding_Dir)
- template<typename Policy >
  Result classify_mpq (const mpq_class &v, bool nan, bool inf, bool sign)
- template<typename Policy >
  bool is_nan_mpq (const mpq_class &v)
- template<typename Policy >
  bool is_minf_mpq (const mpq_class &v)
- template<typename Policy >
  bool is_pinf_mpq (const mpq_class &v)
- template<typename Policy >
  bool is_int_mpq (const mpq_class &v)
- template<typename Policy >
  Result assign_special_mpq (mpq_class &v, Result_Class c, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result construct_mpq_base (mpq_class &to, const From &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result construct_mpq_float (mpq_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpq_float (mpq_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpq_signed_int (mpq_class &to, const From from, Rounding_Dir)

- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpq_unsigned_int (mpq_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result floor_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result ceil_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result trunc_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result neg_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result add_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result sub_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result div_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result idiv_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result rem_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result add_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result sub_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result mul_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result div_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result smod_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result umod_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result abs_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result add_mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result sub_mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result sqrt_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir dir)
- template<typename Policy >
  Result input_mpq (mpq_class &to, std::istream &is, Rounding_Dir dir)
- template<typename Policy >
  Result output_mpq (std::ostream &os, const mpq_class &from, const Numeric_Format &, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpq_long_double (mpq_class &to, const From &from, Rounding_Dir dir)
- template<typename Policy >
  Result round_lt_mpz (mpz_class &to, Rounding_Dir dir)

- template<typename Policy >
  Result round_gt_mpz (mpz_class &to, Rounding_Dir dir)
- mp_size_field_t get_mp_size (const mpz_class &v)
- void set_mp_size (mpz_class &v, mp_size_field_t size)
- template<typename Policy >
  Result classify_mpz (const mpz_class &v, bool nan, bool inf, bool sign)
- template<typename Policy >
  bool is_nan_mpz (const mpz_class &v)
- template<typename Policy >
  bool is_minf_mpz (const mpz_class &v)
- template<typename Policy >
  bool is_pinf_mpz (const mpz_class &v)
- template<typename Policy >
  bool is_int_mpz (const mpz_class &v)
- template<typename Policy >
  Result assign_special_mpz (mpz_class &v, Result_Class c, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  void copy_mpz (mpz_class &to, const mpz_class &from)
- template<typename To_Policy , typename From_Policy , typename From >
  Result construct_mpz_base (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result construct_mpz_float (mpz_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpz_signed_int (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpz_unsigned_int (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpz_float (mpz_class &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result assign_mpz_long_double (mpz_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy >
  Result assign_mpz_mpq (mpz_class &to, const mpq_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy >
  Result neg_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result add_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result sub_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result div_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result idiv_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result rem_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result add_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result sub_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result mul_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)

- template<typename To_Policy , typename From_Policy >
  Result div_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy >
  Result smod_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result umod_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result abs_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result add_mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result sub_mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result gcd_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result gcdext_mpz (mpz_class &to, mpz_class &s, mpz_class &t, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result lcm_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result sqrt_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result_Relation sgn_mp (const Type &x)
- template<typename Policy1 , typename Policy2 , typename Type >
  Result_Relation cmp_mp (const Type &x, const Type &y)
- template<typename Policy >
  Result output_mpz (std::ostream &os, const mpz_class &from, const Numeric_Format &, Rounding_Dir)
- int get_digit (int c, int base=10)

  Returns the integer value associated with the ASCII code `c`*, in the base* `base` *positional number system, if there is such an association; returns* -1 *otherwise.*

- bool sum_sign (bool &a_neg, unsigned long &a_mod, bool b_neg, unsigned long b_mod)

  Adds the number represented (in the modulus-and-sign representation) by `b_neg` *and* `b_mod` *to the number represented by* `a_neg` *and* `a_mod`*, assigning the result to the latter. Returns* `false` *is the result cannot be represented; returns* `true` *otherwise.*

- Result parse_number_part (std::istream &is, number_struct &num)

  Helper function for parse_number(): reads the numerator or denominator part of a number from `is` *into* `num`*, returning the appropriate Result value.*

- Result parse_number (std::istream &is, number_struct &num, number_struct &den)

Variables

- Result_Relation
- const
- From Result
- From Type
- From bool
- From bool Type Type nonconst

- From bool Type Type Result_Class
- From bool Type Type Rounding_Dir To
- From bool Type Type Rounding_Dir From
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir From1
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir From2
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned int
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir To1
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir To2
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir To3
- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type lt (const S &x, const U &y)
- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type le (const S &x, const U &y)
- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type eq (const S &x, const U &y)
- unsigned irrational_precision

  Holds the precision parameter used for irrational calculations.

10.2.1 Detailed Description

Types and functions implementing checked numbers.

10.2.2 Typedef Documentation

10.2.2.1 typedef int16_t Parma_Polyhedra_Library::Checked::int_fast16_t

Definition at line 48 of file checked_int.inlines.hh.

10.2.2.2 typedef int32_t Parma_Polyhedra_Library::Checked::int_fast32_t

Definition at line 52 of file checked_int.inlines.hh.

10.2.2.3 typedef int64_t Parma_Polyhedra_Library::Checked::int_fast64_t

Definition at line 56 of file checked_int.inlines.hh.

10.2.2.4 typedef int Parma_Polyhedra_Library::Checked::mp_size_field_t

This is assumed to be the type of the _mp_size field of GMP's __mpz_struct.

Definition at line 58 of file checked_mpz.inlines.hh.

10.2.2.5 typedef uint16_t Parma_Polyhedra_Library::Checked::uint_fast16_t

Definition at line 60 of file checked_int.inlines.hh.

10.2.2.6 typedef uint32_t Parma_Polyhedra_Library::Checked::uint_fast32_t

Definition at line 64 of file checked_int.inlines.hh.

10.2.2.7 typedef uint64_t Parma_Polyhedra_Library::Checked::uint_fast64_t

Definition at line 68 of file checked_int.inlines.hh.

10.2.3 Function Documentation

10.2.3.1 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::abs_float (Type & *to*, const Type *from*, Rounding_Dir) [inline]

Definition at line 689 of file checked_float.inlines.hh.

10.2.3.2 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::abs_generic (To & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 287 of file checked.inlines.hh.

10.2.3.3 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::abs_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir) [inline]

Definition at line 424 of file checked_mpq.inlines.hh.

10.2.3.4 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::abs_mpz (mpz_class & *to*, const mpz_class & *from*, Rounding_Dir) [inline]

Definition at line 534 of file checked_mpz.inlines.hh.

10.2.3.5 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 602 of file checked_float.inlines.hh.

References Type.

10.2.3.6 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::add_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 343 of file checked_mpq.inlines.hh.

10.2.3.7 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::add_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 449 of file checked_mpz.inlines.hh.

10.2.3.8 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1250 of file checked_int.inlines.hh.

References Type.

10.2.3.9 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1236 of file checked_int.inlines.hh.

References Type.

10.2.3.10 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 437 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.11 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_int_larger (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 930 of file checked_int.inlines.hh.

10.2.3.12 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::add_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 277 of file checked_mpq.inlines.hh.

10.2.3.13 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::add_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 361 of file checked_mpz.inlines.hh.

10.2.3.14 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_mul_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 914 of file checked_float.inlines.hh.

References fma(), fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.15 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_mul_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1458 of file checked_int.inlines.hh.

References Result, and Type.

10.2.3.16 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::add_mul_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 433 of file checked_mpq.inlines.hh.

10.2.3.17 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::add_mul_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 543 of file checked_mpz.inlines.hh.

10.2.3.18 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 977 of file checked_int.inlines.hh.

10.2.3.19 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::add_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 994 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.20 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_exact (To & *to*, const From & *from*, Rounding_Dir) [inline]

Definition at line 274 of file checked.inlines.hh.

10.2.3.21 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_float_float (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 361 of file checked_float.inlines.hh.

References From, and To.

10.2.3.22 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_float_float_exact (To & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 327 of file checked_float.inlines.hh.

10.2.3.23 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_float_float_inexact (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 336 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), From, and limit_precision().

10.2.3.24 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_float_int (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 760 of file checked_float.inlines.hh.

References From.

10.2.3.25 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_float_int_inexact (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 744 of file checked_float.inlines.hh.

References fpu_direct_rounding(), and limit_precision().

10.2.3.26 template<typename To_Policy , typename From_Policy , typename T > Result Parma_Polyhedra_Library::Checked::assign_float_mpq (T & *to*, const mpq_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 843 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

10.2.3.27 template<typename To_Policy , typename From_Policy , typename T > Result Parma_Polyhedra_Library::Checked::assign_float_mpz (T & *to*, const mpz_class & *_from*, Rounding_Dir *dir*) [inline]

Definition at line 803 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

10.2.3.28 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_int_float (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 522 of file checked_int.inlines.hh.

References CHECK_P, From, le, lt, and rint().

10.2.3.29 template<typename To_Policy , typename From_Policy , typename To > Result Parma_Polyhedra_Library::Checked::assign_int_mpq (To & *to*, const mpq_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 699 of file checked_int.inlines.hh.

References Result.

10.2.3.30 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpq_float (mpq_class & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 184 of file checked_mpq.inlines.hh.

10.2.3.31 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpq_long_double (mpq_class & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 515 of file checked_mpq.inlines.hh.

References input_mpq().

10.2.3.32 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpq_signed_int (mpq_class & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 200 of file checked_mpq.inlines.hh.

References From.

10.2.3.33 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpq_unsigned_int (mpq_class & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 221 of file checked_mpq.inlines.hh.

References From.

10.2.3.34 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpz_float (mpz_class & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 263 of file checked_mpz.inlines.hh.

References From, and rint().

10.2.3.35 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpz_long_double (mpz_class & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 295 of file checked_mpz.inlines.hh.

References input_mpq(), and Result.

10.2.3.36 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::assign_mpz_mpq (mpz_class & *to*, const mpq_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 318 of file checked_mpz.inlines.hh.

10.2.3.37 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpz_signed_int (mpz_class & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 231 of file checked_mpz.inlines.hh.

References From.

10.2.3.38 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::assign_mpz_unsigned_int (mpz_class & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 251 of file checked_mpz.inlines.hh.

References From.

10.2.3.39 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::assign_nan (Type & *to*, Result *r*) [inline]

Definition at line 624 of file checked.inlines.hh.

10.2.3.40 template<typename To_Policy , typename From_Policy , typename To > Result Parma_Polyhedra_Library::Checked::assign_signed_int_mpz (To & *to*, const mpz_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 608 of file checked_int.inlines.hh.

References PPL_GT_SILENT, PPL_LT_SILENT, and To.

10.2.3.41 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_signed_int_signed_int (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 352 of file checked_int.inlines.hh.

References CHECK_P, From, PPL_GT_SILENT, PPL_LT_SILENT, and To.

10.2.3.42 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_signed_int_unsigned_int (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 370 of file checked_int.inlines.hh.

References CHECK_P, From, and To.

10.2.3.43 template<typename Policy , typename T > Result Parma_Polyhedra_Library::Checked::assign_special_float (T & *v*, Result_Class *c*, Rounding_Dir) [inline]

Definition at line 206 of file checked_float.inlines.hh.

References PPL_NAN.

10.2.3.44 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::assign_special_int (Type & *v*, Result_Class *c*, Rounding_Dir *dir*) [inline]

Definition at line 304 of file checked_int.inlines.hh.

10.2.3.45 template<typename Policy > Result Parma_Polyhedra_Library::Checked::assign_special_mpq (mpq_class & *v*, Result_Class *c*, Rounding_Dir) [inline]

Definition at line 103 of file checked_mpq.inlines.hh.

10.2.3.46 template<typename Policy > Result Parma_Polyhedra_Library::Checked::assign_special_mpz (mpz_class & *v*, Result_Class *c*, Rounding_Dir) [inline]

Definition at line 133 of file checked_mpz.inlines.hh.

References set_mp_size().

10.2.3.47 template<typename To_Policy , typename From_Policy , typename To > Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_mpz (To & *to*, const mpz_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 658 of file checked_int.inlines.hh.

References CHECK_P, PPL_GT_SILENT, and To.

10.2.3.48 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_signed_int (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 382 of file checked_int.inlines.hh.

References CHECK_P, From, and To.

10.2.3.49 template<typename To_Policy , typename From_Policy , typename To , typename From > Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_unsigned_int (To & *to*, const From *from*, Rounding_Dir *dir*) [inline]

Definition at line 396 of file checked_int.inlines.hh.

References CHECK_P, From, PPL_GT_SILENT, and To.

10.2.3.50 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::ceil_float (Type & *to*, const Type *from*, Rounding_Dir) [inline]

Definition at line 393 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), limit_precision(), and rint().

10.2.3.51 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::ceil_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir) [inline]

Definition at line 246 of file checked_mpq.inlines.hh.

10.2.3.52 template<typename Policy , typename T > Result Parma_Polyhedra_Library::Checked::classify_float (const T *v*, bool *nan*, bool *inf*, bool *sign*) [inline]

Definition at line 151 of file checked_float.inlines.hh.

References CHECK_P.

10.2.3.53 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::classify_int (const Type *v*, bool *nan*, bool *inf*, bool *sign*) [inline]

Definition at line 193 of file checked_int.inlines.hh.

10.2.3.54 template<typename Policy > Result Parma_Polyhedra_Library::Checked::classify_mpq (const mpq_class & *v*, bool *nan*, bool *inf*, bool *sign*) [inline]

Definition at line 37 of file checked_mpq.inlines.hh.

References Result.

10.2.3.55 template<typename Policy > Result Parma_Polyhedra_Library::Checked::classify_mpz (const mpz_class & *v*, bool *nan*, bool *inf*, bool *sign*) [inline]

Definition at line 73 of file checked_mpz.inlines.hh.

References get_mp_size(), and Result.

10.2.3.56 template<typename Policy1 , typename Policy2 , typename Type > Result_Relation Parma_Polyhedra_Library::Checked::cmp_float (const Type *x*, const Type *y*) [inline]

Definition at line 732 of file checked_float.inlines.hh.

10.2.3.57 template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 > Result_Relation Parma_Polyhedra_Library::Checked::cmp_generic (const Type1 & *x*, const Type2 & *y*) [inline]

Definition at line 614 of file checked.inlines.hh.

10.2.3.58 template<typename Policy1 , typename Policy2 , typename Type > Result_Relation Parma_Polyhedra_Library::Checked::cmp_mp (const Type & *x*, const Type & *y*) [inline]

Definition at line 622 of file checked_mpz.inlines.hh.

References Parma_Polyhedra_Library::cmp().

10.2.3.59 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::construct_mpq_base (mpq_class & *to*, const From & *from*, Rounding_Dir) [inline]

Definition at line 138 of file checked_mpq.inlines.hh.

10.2.3.60 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::construct_mpq_float (mpq_class & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 156 of file checked_mpq.inlines.hh.

10.2.3.61 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::construct_mpz_base (mpz_class & *to*, const From *from*, Rounding_Dir) [inline]

Definition at line 177 of file checked_mpz.inlines.hh.

10.2.3.62 template<typename To_Policy , typename From_Policy , typename From > Result Parma_Polyhedra_Library::Checked::construct_mpz_float (mpz_class & *to*, const From & *from*, Rounding_Dir *dir*) [inline]

Definition at line 194 of file checked_mpz.inlines.hh.

References From, and rint().

10.2.3.63 template<typename To_Policy , typename From_Policy , typename Type > Enable_If<Is_Same<To_Policy, From_Policy>::value, void>::type Parma_Polyhedra_Library::Checked::copy_generic (Type & *to*, const Type & *from*) [inline]

Definition at line 281 of file checked.inlines.hh.

10.2.3.64 template<typename To_Policy , typename From_Policy > void Parma_Polyhedra_Library::Checked::copy_mpz (mpz_class & *to*, const mpz_class & *from*) [inline]

Definition at line 161 of file checked_mpz.inlines.hh.

References get_mp_size(), and set_mp_size().

10.2.3.65 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 641 of file checked_float.inlines.hh.

References Type.

10.2.3.66 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::div_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 381 of file checked_mpq.inlines.hh.

10.2.3.67 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::div_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 485 of file checked_mpz.inlines.hh.

10.2.3.68 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1191 of file checked_int.inlines.hh.

References Type.

10.2.3.69 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1170 of file checked_int.inlines.hh.

References Type.

10.2.3.70 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 529 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.71 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::div_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 304 of file checked_mpq.inlines.hh.

References CHECK_P.

10.2.3.72 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::div_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir *dir*) [inline]

Definition at line 388 of file checked_mpz.inlines.hh.

References CHECK_P.

10.2.3.73 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1093 of file checked_int.inlines.hh.

References CHECK_P, and Type.

10.2.3.74 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::div_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1113 of file checked_int.inlines.hh.

References CHECK_P, and Type.

10.2.3.75 template<typename T1 , typename T2 > Enable_If<(Safe_Int_Comparison<T1, T2>::value || Safe_Conversion<T1, T2>::value || Safe_Conversion<T2, T1>::value), bool>::type Parma_Polyhedra_Library::Checked::eq (const T1 & *x*, const T2 & *y*) [inline]

Definition at line 476 of file checked.inlines.hh.

10.2.3.76 template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 > bool Parma_Polyhedra_Library::Checked::eq_p (const Type1 & *x*, const Type2 & *y*) [inline]

Definition at line 607 of file checked.inlines.hh.

10.2.3.77 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::floor_float (Type & *to*, const Type *from*, Rounding_Dir) [inline]

Definition at line 370 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), limit_precision(), and rint().

10.2.3.78 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::floor_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir) [inline]

Definition at line 235 of file checked_mpq.inlines.hh.

10.2.3.79 long double Parma_Polyhedra_Library::Checked::fma (long double *x*, long double *y*, long double *z*) [inline]

Definition at line 57 of file checked_float.inlines.hh.

10.2.3.80 double Parma_Polyhedra_Library::Checked::fma (double *x*, double *y*, double *z*) [inline]

Definition at line 47 of file checked_float.inlines.hh.

References fma().

10.2.3.81 float Parma_Polyhedra_Library::Checked::fma (float *x*, float *y*, float *z*) [inline]

Definition at line 37 of file checked_float.inlines.hh.

Referenced by add_mul_float(), fma(), and sub_mul_float().

10.2.3.82 bool Parma_Polyhedra_Library::Checked::fpu_direct_rounding (Rounding_Dir *dir*) [inline]

Definition at line 102 of file checked_float.inlines.hh.

Referenced by add_float(), add_mul_float(), assign_float_float_inexact(), assign_float_int_inexact(), ceil_float(), div_float(), floor_float(), mul_float(), sqrt_float(), sub_float(), and sub_mul_float().

10.2.3.83 bool Parma_Polyhedra_Library::Checked::fpu_inverse_rounding (Rounding_Dir *dir*) [inline]

Definition at line 107 of file checked_float.inlines.hh.

Referenced by add_float(), add_mul_float(), assign_float_float_inexact(), ceil_float(), div_float(), floor_float(), mul_float(), sub_float(), and sub_mul_float().

10.2.3.84 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 > Result Parma_Polyhedra_Library::Checked::gcd_exact (To & *to*, const From1 & *x*, const From2 & *y*, Rounding_Dir *dir*) [inline]

Definition at line 315 of file checked.inlines.hh.

10.2.3.85 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From > void Parma_Polyhedra_Library::Checked::gcd_exact_noabs (To & *to*, const From & *x*, const From & *y*) [inline]

Definition at line 297 of file checked.inlines.hh.

References To.

10.2.3.86 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::gcd_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 563 of file checked_mpz.inlines.hh.

10.2.3.87 template<typename To1_Policy , typename To2_Policy , typename To3_Policy , typename From1_Policy , typename From2_Policy , typename To1 , typename To2 , typename To3 , typename From1 , typename From2 > Result Parma_Polyhedra_Library::Checked::gcdext_exact (To1 & *to*, To2 & *s*, To3 & *t*, const From1 & *x*, const From2 & *y*, Rounding_Dir *dir*) [inline]

Definition at line 325 of file checked.inlines.hh.

References From2, Result, To1, To2, and To3.

10.2.3.88 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::gcdext_mpz (mpz_class & *to*, mpz_class & *s*, mpz_class & *t*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 572 of file checked_mpz.inlines.hh.

10.2.3.89 int Parma_Polyhedra_Library::Checked::get_digit (int *c*, int *base* = 10) [inline]

Returns the integer value associated with the ASCII code `c`, in the base `base` positional number system, if there is such an association; returns -1 otherwise.

Definition at line 54 of file checked.cc.

Referenced by parse_number_part().

10.2.3.90 mp_size_field_t Parma_Polyhedra_Library::Checked::get_mp_size (const mpz_class & *v*) [inline]

Definition at line 62 of file checked_mpz.inlines.hh.

Referenced by classify_mpz(), copy_mpz(), is_minf_mpz(), is_nan_mpz(), and is_pinf_mpz().

10.2.3.91 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::idiv_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 562 of file checked_float.inlines.hh.

References Result, and Type.

10.2.3.92 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::idiv_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir *dir*) [inline]

Definition at line 316 of file checked_mpq.inlines.hh.

References CHECK_P.

10.2.3.93 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::idiv_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 422 of file checked_mpz.inlines.hh.

References CHECK_P.

10.2.3.94 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::idiv_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1128 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.95 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::idiv_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir) [inline]

Definition at line 1140 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.96 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::input_generic (Type & *to*, std::istream & *is*, Rounding_Dir *dir*) [inline]

Definition at line 631 of file checked.inlines.hh.

10.2.3.97 template<typename Policy > Result Parma_Polyhedra_Library::Checked::input_mpq (mpq_class & *to*, std::istream & *is*, Rounding_Dir *dir*) [inline]

Definition at line 485 of file checked_mpq.inlines.hh.

References input_mpq(), Result, and Result_Class.

10.2.3.98 Result Parma_Polyhedra_Library::Checked::input_mpq (mpq_class & *to*, std::istream & *is*)

Definition at line 336 of file checked.cc.

References Parma_Polyhedra_Library::Checked::number_struct::base, Parma_Polyhedra_Library::Checked::number_struct::base_for_exponent, Parma_Polyhedra_Library::Checked::number_struct::exponent, Parma_Polyhedra_Library::Checked::number_struct::mantissa, Parma_Polyhedra_Library::Checked::number_struct::neg_exponent, Parma_Polyhedra_Library::Checked::number_struct::neg_mantissa, parse_number(), and Result.

Referenced by assign_mpq_long_double(), assign_mpz_long_double(), and input_mpq().

10.2.3.99 template<typename Policy , typename T > int Parma_Polyhedra_Library::Checked::is_inf_float (const T *v*) [inline]

Definition at line 181 of file checked_float.inlines.hh.

References CHECK_P.

10.2.3.100 template<typename Policy , typename T > bool Parma_Polyhedra_Library::Checked::is_int_float (const T *v*) [inline]

Definition at line 200 of file checked_float.inlines.hh.

References rint().

10.2.3.101 template<typename Policy , typename Type > bool Parma_Polyhedra_Library::Checked::is_int_int (const Type *v*) [inline]

Definition at line 286 of file checked_int.inlines.hh.

10.2.3.102 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_int_mpq (const mpq_class & *v*) [inline]

Definition at line 91 of file checked_mpq.inlines.hh.

10.2.3.103 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_int_mpz (const mpz_class & *v*) [inline]

Definition at line 125 of file checked_mpz.inlines.hh.

10.2.3.104 template<typename Policy , typename T > bool Parma_Polyhedra_Library::Checked::is_minf_float (const T *v*) [inline]

Definition at line 187 of file checked_float.inlines.hh.

10.2.3.105 template<typename Policy , typename Type > bool Parma_Polyhedra_Library::Checked::is_minf_int (const Type *v*) [inline]

Definition at line 248 of file checked_int.inlines.hh.

10.2.3.106 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_minf_mpq (const mpq_class & *v*) [inline]

Definition at line 71 of file checked_mpq.inlines.hh.

10.2.3.107 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_minf_mpz (const mpz_class & *v*) [inline]

Definition at line 107 of file checked_mpz.inlines.hh.

References get_mp_size().

10.2.3.108 template<typename Policy , typename T > bool Parma_Polyhedra_Library::Checked::is_nan_float (const T *v*) [inline]

Definition at line 174 of file checked_float.inlines.hh.

References CHECK_P.

10.2.3.109 template<typename Policy , typename Type > bool Parma_Polyhedra_Library::Checked::is_nan_int (const Type *v*) [inline]

Definition at line 230 of file checked_int.inlines.hh.

10.2.3.110 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_nan_mpq (const mpq_class & *v*) [inline]

Definition at line 61 of file checked_mpq.inlines.hh.

10.2.3.111 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_nan_mpz (const mpz_class & *v*) [inline]

Definition at line 98 of file checked_mpz.inlines.hh.

References get_mp_size().

10.2.3.112 template<typename Policy , typename T > bool Parma_Polyhedra_Library::Checked::is_pinf_float (const T *v*) [inline]

Definition at line 193 of file checked_float.inlines.hh.

10.2.3.113 template<typename Policy , typename Type > bool Parma_Polyhedra_Library::Checked::is_pinf_int (const Type *v*) [inline]

Definition at line 267 of file checked_int.inlines.hh.

10.2.3.114 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_pinf_mpq (const mpq_class & *v*) [inline]

Definition at line 81 of file checked_mpq.inlines.hh.

10.2.3.115 template<typename Policy > bool Parma_Polyhedra_Library::Checked::is_pinf_mpz (const mpz_class & *v*) [inline]

Definition at line 116 of file checked_mpz.inlines.hh.

References get_mp_size().

10.2.3.116 template<typename Type > void Parma_Polyhedra_Library::Checked::isqrtrem (Type & *q*, Type & *r*, const Type *from*) [inline]

Definition at line 1421 of file checked_int.inlines.hh.

References Type.

Referenced by sqrt_unsigned_int().

10.2.3.117 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 > Result Parma_Polyhedra_Library::Checked::lcm_gcd_exact (To & *to*, const From1 & *x*, const From2 & *y*, Rounding_Dir *dir*) [inline]

Definition at line 410 of file checked.inlines.hh.

References Result, and To.

10.2.3.118 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::lcm_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 584 of file checked_mpz.inlines.hh.

10.2.3.119 template<typename T1 , typename T2 > Enable_If<(Safe_Int_Comparison<T1, T2>::value || Safe_Conversion<T1, T2>::value || Safe_Conversion<T2, T1>::value), bool>::type Parma_Polyhedra_Library::Checked::le (const T1 & *x*, const T2 & *y*) [inline]

Definition at line 469 of file checked.inlines.hh.

10.2.3.120 template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 > bool Parma_Polyhedra_Library::Checked::le_p (const Type1 & *x*, const Type2 & *y*) [inline]

Definition at line 600 of file checked.inlines.hh.

10.2.3.121 void Parma_Polyhedra_Library::Checked::limit_precision (const long double &) [inline]

Definition at line 146 of file checked_float.inlines.hh.

10.2.3.122 void Parma_Polyhedra_Library::Checked::limit_precision (const double & *v*) [inline]

Definition at line 141 of file checked_float.inlines.hh.

10.2.3.123 void Parma_Polyhedra_Library::Checked::limit_precision (const float & *v*) [inline]

Definition at line 136 of file checked_float.inlines.hh.

Referenced by add_float(), add_mul_float(), assign_float_float_inexact(), assign_float_int_inexact(), ceil_float(), div_float(), floor_float(), mul_float(), sqrt_float(), sub_float(), and sub_mul_float().

10.2.3.124 template<typename T1 , typename T2 > Enable_If<(Safe_Int_Comparison<T1, T2>::value || Safe_Conversion<T1, T2>::value || Safe_Conversion<T2, T1>::value), bool>::type Parma_Polyhedra_Library::Checked::lt (const T1 & *x*, const T2 & *y*) [inline]

Definition at line 462 of file checked.inlines.hh.

10.2.3.125 template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 > bool Parma_Polyhedra_Library::Checked::lt_p (const Type1 & *x*, const Type2 & *y*) [inline]

Definition at line 593 of file checked.inlines.hh.

10.2.3.126 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 628 of file checked_float.inlines.hh.

References Type.

10.2.3.127 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::mul_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 369 of file checked_mpq.inlines.hh.

10.2.3.128 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::mul_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 475 of file checked_mpz.inlines.hh.

10.2.3.129 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1328 of file checked_int.inlines.hh.

References PPL_GT_SILENT, PPL_LT_SILENT, and Type.

10.2.3.130 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1304 of file checked_int.inlines.hh.

References PPL_GT_SILENT, and Type.

10.2.3.131 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 497 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.132 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_int_larger (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 946 of file checked_int.inlines.hh.

10.2.3.133 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::mul_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 295 of file checked_mpq.inlines.hh.

10.2.3.134 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::mul_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 379 of file checked_mpz.inlines.hh.

10.2.3.135 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1035 of file checked_int.inlines.hh.

10.2.3.136 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::mul_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1074 of file checked_int.inlines.hh.

10.2.3.137 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::neg_float (Type & *to*, const Type *from*, Rounding_Dir) [inline]

Definition at line 427 of file checked_float.inlines.hh.

10.2.3.138 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::neg_int_larger (Type & *to*, const Type *x*, Rounding_Dir *dir*) [inline]

Definition at line 922 of file checked_int.inlines.hh.

10.2.3.139 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::neg_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir) [inline]

Definition at line 268 of file checked_mpq.inlines.hh.

10.2.3.140 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::neg_mpz (mpz_class & *to*, const mpz_class & *from*, Rounding_Dir) [inline]

Definition at line 352 of file checked_mpz.inlines.hh.

10.2.3.141 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::neg_signed_int (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 954 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.142 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::neg_unsigned_int (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 966 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.143 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::output_char (std::ostream & *os*, Type & *from*, const Numeric_Format &, Rounding_Dir) [inline]

Definition at line 1502 of file checked_int.inlines.hh.

References int.

10.2.3.144 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::output_float (std::ostream & *os*, const Type *from*, const Numeric_Format &, Rounding_Dir) [inline]

Definition at line 980 of file checked_float.inlines.hh.

10.2.3.145 template<typename Policy , typename Type > Result Parma_Polyhedra_Library::Checked::output_int (std::ostream & *os*, Type & *from*, const Numeric_Format &, Rounding_Dir) [inline]

Definition at line 1510 of file checked_int.inlines.hh.

10.2.3.146 template<typename Policy > Result Parma_Polyhedra_Library::Checked::output_mpq (std::ostream & *os*, const mpq_class & *from*, const Numeric_Format &, Rounding_Dir) [inline]

Definition at line 503 of file checked_mpq.inlines.hh.

10.2.3.147 template<typename Policy > Result Parma_Polyhedra_Library::Checked::output_mpz (std::ostream & *os*, const mpz_class & *from*, const Numeric_Format &, Rounding_Dir) [inline]

Definition at line 632 of file checked_mpz.inlines.hh.

10.2.3.148 Result Parma_Polyhedra_Library::Checked::parse_number (std::istream & *is*, number_struct & *num*, number_struct & *den*)

Definition at line 304 of file checked.cc.

References Parma_Polyhedra_Library::Checked::number_struct::base, Parma_Polyhedra_Library::Checked::number_struct::base_for_exponent, Parma_Polyhedra_Library::Checked::number_struct::exponent, Parma_Polyhedra_Library::Checked::number_struct::neg_exponent, parse_number_part(), Result, and sum_sign().

Referenced by input_mpq().

10.2.3.149 Result Parma_Polyhedra_Library::Checked::parse_number_part (std::istream & *is*, number_struct & *num*)

Helper function for parse_number(): reads the numerator or denominator part of a number from `is` into `num`, returning the appropriate Result value.

Definition at line 98 of file checked.cc.

References Parma_Polyhedra_Library::Checked::number_struct::base, Parma_Polyhedra_Library::Checked::number_struct::base_for_exponent, Parma_Polyhedra_Library::Checked::number_struct::exponent, get_digit(), Parma_Polyhedra_Library::Checked::number_struct::mantissa, Parma_Polyhedra_Library::Checked::number_struct::neg_exponent, Parma_Polyhedra_Library::Checked::number_struct::neg_mantissa, and sum_sign().

Referenced by parse_number().

10.2.3.150 From bool Type Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_0_0 (is_pinf, bool, const, Type)

Type Constraints

10.2.3.151 From bool Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_0_0 (is_nan, bool, const, Type)

Type Constraints

10.2.3.152 From bool Type Type Parma_Polyhedra_Library::Checked::PPL_DECLARE_-FUN1_0_2 (assign_special, Result, nonconst, Type, Result_Class, Rounding_Dir)

Type Constraints

10.2.3.153 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_-FUN1_1_2 (output, Result, std::ostream &, const, Type, const Numeric_Format &, Rounding_Dir)

Type Constraints

10.2.3.154 From Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_0 (cmp, Result_Relation, const, Type1, const, Type2)

Type Constraints

10.2.3.155 Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_0 (copy, void, nonconst, Type1, const, Type2)

10.2.3.156 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_1 (abs, Result, nonconst, To, const, From, Rounding_Dir)

Type Constraints

10.2.3.157 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_1 (trunc, Result, nonconst, To, const, From, Rounding_Dir)

Type Constraints

10.2.3.158 From bool Type Type Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_1 (floor, Result, nonconst, To, const, From, Rounding_Dir)

Type Constraints

10.2.3.159 From bool Type Type Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_1 (construct, Result, nonconst, To, const, From, Rounding_Dir)

Type Constraints

10.2.3.160 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (smod_2exp, Result, nonconst, To, const, From, unsigned *int*, Rounding_Dir)

Type Constraints

10.2.3.161 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (mul_2exp, Result, nonconst, To, const, From, unsigned *int*, Rounding_Dir)

Type Constraints

10.2.3.162 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (add_2exp, Result, nonconst, To, const, From, unsigned *int*, Rounding_Dir)

Type Constraints

10.2.3.163 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (lcm, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.164 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (gcd, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.165 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (add_mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.166 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (rem, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.167 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.168 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (add, Result, nonconst, To, const, From1, const, From2, Rounding_Dir)

Type Constraints

10.2.3.169 template<typename To_Policy , typename To > struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct_special) [read]

Definition at line 265 of file checked.inlines.hh.

References Result, Result_Class, and To.

10.2.3.170 template<typename To_Policy , typename From_Policy , typename To , typename From > struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct) [read]

Type Constraints

Definition at line 257 of file checked.inlines.hh.

References From, Result, and To.

10.2.3.171 template<typename Policy , typename Type > struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct) [read]

10.2.3.172 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, double)

10.2.3.173 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, float)

10.2.3.174 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, unsigned *char*)

10.2.3.175 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, signed *char*)

10.2.3.176 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, char)

10.2.3.177 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, unsigned *char*)

10.2.3.178 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, signed *char*)

10.2.3.179 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, char)

10.2.3.180 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (long *double*, double)

10.2.3.181 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (long *double*, float)

10.2.3.182 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (double, float)

10.2.3.183 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long *long*, unsigned *char*)

10.2.3.184 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long *long*, char)

10.2.3.185 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned *long*, unsigned *char*)

10.2.3.186 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned *long*, char)

10.2.3.187 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned *short*, unsigned *char*)

10.2.3.188 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned *short*, char)

10.2.3.189 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed long *long*, signed *short*)

10.2.3.190 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed long *long*, signed *char*)

10.2.3.191 Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed *short*, signed *char*)

10.2.3.192 template<typename T > void Parma_Polyhedra_Library::Checked::pred_float (T & *v*) [inline]

Definition at line 225 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

Referenced by round_lt_float().

10.2.3.193 template<typename Policy > void Parma_Polyhedra_Library::Checked::prepare_inexact (Rounding_Dir *dir*) [inline]

Definition at line 284 of file checked_float.inlines.hh.

10.2.3.194 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::rem_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir) [inline]

Definition at line 582 of file checked_float.inlines.hh.

10.2.3.195 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::rem_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 328 of file checked_mpq.inlines.hh.

References CHECK_P.

10.2.3.196 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::rem_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 437 of file checked_mpz.inlines.hh.

References CHECK_P.

10.2.3.197 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::rem_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir) [inline]

Definition at line 1150 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.198 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::rem_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir) [inline]

Definition at line 1160 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.199 template<typename Policy > Result Parma_Polyhedra_Library::Checked::result_relation (Rounding_Dir *dir*) [inline]

Definition at line 292 of file checked_float.inlines.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::ascii_load(), Parma_Polyhedra_Library::DB_Matrix< T >::ascii_load(), and Parma_Polyhedra_Library::check_result().

10.2.3.200 long double Parma_Polyhedra_Library::Checked::rint (long double *x*) [inline]

Definition at line 87 of file checked_float.inlines.hh.

References rint().

10.2.3.201 double Parma_Polyhedra_Library::Checked::rint (double *x*) [inline]

Definition at line 74 of file checked_float.inlines.hh.

Referenced by assign_int_float(), assign_mpz_float(), ceil_float(), construct_mpz_float(), floor_float(), is_int_float(), and rint().

10.2.3.202 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir To Result Parma_Polyhedra_Library::Checked::round (To & *to*, Result *r*, Rounding_Dir *dir*)

10.2.3.203 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_gt_float (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 273 of file checked_float.inlines.hh.

References succ_float().

10.2.3.204 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_gt_int (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 162 of file checked_int.inlines.hh.

10.2.3.205 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_gt_int_no_overflow (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 133 of file checked_int.inlines.hh.

10.2.3.206 template<typename Policy > Result Parma_Polyhedra_Library::Checked::round_gt_mpz (mpz_class & *to*, Rounding_Dir *dir*) [inline]

Definition at line 45 of file checked_mpz.inlines.hh.

10.2.3.207 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_lt_float (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 263 of file checked_float.inlines.hh.

References pred_float().

10.2.3.208 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_lt_int (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 143 of file checked_int.inlines.hh.

10.2.3.209 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::round_lt_int_no_overflow (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 123 of file checked_int.inlines.hh.

10.2.3.210 template<typename Policy > Result Parma_Polyhedra_Library::Checked::round_lt_mpz (mpz_class & *to*, Rounding_Dir *dir*) [inline]

Definition at line 35 of file checked_mpz.inlines.hh.

10.2.3.211 void Parma_Polyhedra_Library::Checked::set_mp_size (mpz_class & *v*, mp_size_field_t *size*) [inline]

Definition at line 67 of file checked_mpz.inlines.hh.

Referenced by assign_special_mpz(), and copy_mpz().

10.2.3.212 template<typename Policy , typename T > Result Parma_Polyhedra_Library::Checked::set_neg_overflow_float (T & *to*, Rounding_Dir *dir*) [inline]

Definition at line 769 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

10.2.3.213 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::set_neg_overflow_int (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 91 of file checked_int.inlines.hh.

10.2.3.214 template<typename Policy , typename T > Result Parma_Polyhedra_Library::Checked::set_pos_overflow_float (T & *to*, Rounding_Dir *dir*) [inline]

Definition at line 786 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

10.2.3.215 template<typename Policy , typename To > Result Parma_Polyhedra_Library::Checked::set_pos_overflow_int (To & *to*, Rounding_Dir *dir*) [inline]

Definition at line 107 of file checked_int.inlines.hh.

10.2.3.216 template<typename Policy , typename Type > Result_Relation Parma_Polyhedra_Library::Checked::sgn_float (const Type *x*) [inline]

Definition at line 720 of file checked_float.inlines.hh.

10.2.3.217 template<typename Policy , typename Type > Result_Relation Parma_Polyhedra_Library::Checked::sgn_generic (const Type & *x*) [inline]

Definition at line 434 of file checked.inlines.hh.

10.2.3.218 template<typename Policy , typename Type > Result_Relation Parma_Polyhedra_Library::Checked::sgn_mp (const Type & *x*) [inline]

Definition at line 612 of file checked_mpz.inlines.hh.

10.2.3.219 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::smod_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 654 of file checked_float.inlines.hh.

References Type.

10.2.3.220 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::smod_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 393 of file checked_mpq.inlines.hh.

10.2.3.221 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::smod_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 511 of file checked_mpz.inlines.hh.

10.2.3.222 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::smod_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 1383 of file checked_int.inlines.hh.

References Type.

10.2.3.223 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::smod_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1367 of file checked_int.inlines.hh.

References Type.

10.2.3.224 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sqrt_float (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 698 of file checked_float.inlines.hh.

References fpu_direct_rounding(), and limit_precision().

10.2.3.225 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::sqrt_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 455 of file checked_mpq.inlines.hh.

References CHECK_P, and Result.

10.2.3.226 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::sqrt_mpz (mpz_class & *to*, const mpz_class & *from*, Rounding_Dir *dir*) [inline]

Definition at line 593 of file checked_mpz.inlines.hh.

References CHECK_P.

10.2.3.227 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sqrt_signed_int (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 1449 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.228 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sqrt_unsigned_int (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 1437 of file checked_int.inlines.hh.

References isqrtrem(), and Type.

10.2.3.229 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 615 of file checked_float.inlines.hh.

References Type.

10.2.3.230 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::sub_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 356 of file checked_mpq.inlines.hh.

10.2.3.231 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::sub_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 462 of file checked_mpz.inlines.hh.

10.2.3.232 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1284 of file checked_int.inlines.hh.

References Type.

10.2.3.233 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1270 of file checked_int.inlines.hh.

References Type.

10.2.3.234 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 467 of file checked_float.inlines.hh.

References fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.235 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_int_larger (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 938 of file checked_int.inlines.hh.

10.2.3.236 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::sub_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 286 of file checked_mpq.inlines.hh.

10.2.3.237 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::sub_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 370 of file checked_mpz.inlines.hh.

10.2.3.238 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_mul_float (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 947 of file checked_float.inlines.hh.

References fma(), fpu_direct_rounding(), fpu_inverse_rounding(), and limit_precision().

10.2.3.239 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_mul_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1480 of file checked_int.inlines.hh.

References Result, and Type.

10.2.3.240 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::sub_mul_mpq (mpq_class & *to*, const mpq_class & *x*, const mpq_class & *y*, Rounding_Dir) [inline]

Definition at line 443 of file checked_mpq.inlines.hh.

10.2.3.241 template<typename To_Policy , typename From1_Policy , typename From2_Policy > Result Parma_Polyhedra_Library::Checked::sub_mul_mpz (mpz_class & *to*, const mpz_class & *x*, const mpz_class & *y*, Rounding_Dir) [inline]

Definition at line 553 of file checked_mpz.inlines.hh.

10.2.3.242 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_signed_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1006 of file checked_int.inlines.hh.

10.2.3.243 template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::sub_unsigned_int (Type & *to*, const Type *x*, const Type *y*, Rounding_Dir *dir*) [inline]

Definition at line 1023 of file checked_int.inlines.hh.

References CHECK_P.

10.2.3.244 template<typename T > void Parma_Polyhedra_Library::Checked::succ_float (T & *v*) [inline]

Definition at line 244 of file checked_float.inlines.hh.

References Parma_Polyhedra_Library::Bool< false >::value.

Referenced by round_gt_float().

10.2.3.245 bool Parma_Polyhedra_Library::Checked::sum_sign (bool & *a_neg*, unsigned long & *a_mod*, bool *b_neg*, unsigned long *b_mod*) [inline]

Adds the number represented (in the modulus-and-sign representation) by `b_neg` and `b_mod` to the number represented by `a_neg` and `a_mod`, assigning the result to the latter. Returns `false` is the result cannot be represented; returns `true` otherwise.

Definition at line 75 of file checked.cc.

Referenced by parse_number(), and parse_number_part().

10.2.3.246 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::trunc_float (Type & *to*, const Type *from*, Rounding_Dir *dir*) [inline]

Definition at line 416 of file checked_float.inlines.hh.

10.2.3.247 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::trunc_mpq (mpq_class & *to*, const mpq_class & *from*, Rounding_Dir) [inline]

Definition at line 257 of file checked_mpq.inlines.hh.

10.2.3.248 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::umod_2exp_float (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 673 of file checked_float.inlines.hh.

References Type.

10.2.3.249 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::umod_2exp_mpq (mpq_class & *to*, const mpq_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 411 of file checked_mpq.inlines.hh.

10.2.3.250 template<typename To_Policy , typename From_Policy > Result Parma_Polyhedra_Library::Checked::umod_2exp_mpz (mpz_class & *to*, const mpz_class & *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 524 of file checked_mpz.inlines.hh.

10.2.3.251 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::umod_2exp_signed_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir *dir*) [inline]

Definition at line 1407 of file checked_int.inlines.hh.

References Type.

10.2.3.252 template<typename To_Policy , typename From_Policy , typename Type > Result Parma_Polyhedra_Library::Checked::umod_2exp_unsigned_int (Type & *to*, const Type *x*, unsigned int *exp*, Rounding_Dir) [inline]

Definition at line 1396 of file checked_int.inlines.hh.

References Type.

10.2.4 Variable Documentation

10.2.4.1 From bool Type Parma_Polyhedra_Library::Checked::bool

Definition at line 434 of file checked.defs.hh.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::external_memory_in_bytes().

10.2.4.2 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::const

Definition at line 430 of file checked.defs.hh.

10.2.4.3 Enable_If<(!Safe_Conversion< T1, T2 >::value &&!Safe_Conversion< T2, T1 >::value &&(!C_Integer< T1 >::value||!C_Integer< T2 >::value)), bool >::type Parma_Polyhedra_Library::Checked::eq [inline]

Definition at line 516 of file checked.inlines.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_singleton(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

10.2.4.4 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::From

Definition at line 450 of file checked.defs.hh.

Referenced by assign_float_float(), assign_float_float_inexact(), assign_float_int(), assign_int_float(), assign_mpq_signed_int(), assign_mpq_unsigned_int(), assign_mpz_float(), assign_mpz_signed_int(), assign_mpz_unsigned_int(), assign_signed_int_signed_int(), assign_signed_int_unsigned_int(), assign_unsigned_int_signed_int(), assign_unsigned_int_unsigned_int(), construct_mpz_float(), Parma_Polyhedra_Library::neg_assign(), and PPL_FUNCTION_CLASS().

10.2.4.5 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::From1

Definition at line 467 of file checked.defs.hh.

10.2.4.6 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::From2

Definition at line 467 of file checked.defs.hh.

Referenced by gcdext_exact().

10.2.4.7 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Parma_Polyhedra_Library::Checked::int

Definition at line 485 of file checked.defs.hh.

Referenced by output_char().

10.2.4.8 unsigned Parma_Polyhedra_Library::Checked::irrational_precision

Holds the precision parameter used for irrational calculations.

Definition at line 37 of file checked.cc.

Referenced by Parma_Polyhedra_Library::irrational_precision(), and Parma_Polyhedra_Library::set_irrational_precision().

10.2.4.9 Enable_If<(!Safe_Conversion< T1, T2 >::value &&!Safe_Conversion< T2, T1 >::value &&(!C_Integer< T1 >::value||!C_Integer< T2 >::value)), bool >::type Parma_Polyhedra_Library::Checked::le [inline]

Definition at line 500 of file checked.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), assign_int_float(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition_aux(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruence(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), Parma_Polyhedra_Library::Grid::simplify_using_context_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

10.2.4.10 Enable_If<(!Safe_Conversion< T1, T2 >::value &&!Safe_Conversion< T2, T1 >::value &&(!C_Integer< T1 >::value||!C_Integer< T2 >::value)), bool >::type Parma_Polyhedra_Library::Checked::lt [inline]

Definition at line 484 of file checked.inlines.hh.

Referenced by assign_int_float(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains().

10.2.4.11 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::nonconst

Definition at line 446 of file checked.defs.hh.

10.2.4.12 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::Result

Definition at line 434 of file checked.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), add_mul_int(), Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::addmod(), Parma_Polyhedra_Library::OR_Matrix< T >::ascii_load(), Parma_Polyhedra_Library::DB_Matrix< T >::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), assign_int_mpq(), assign_mpz_long_double(), Parma_Polyhedra_Library::assign_rem(), classify_mpq(), classify_mpz(), Parma_Polyhedra_Library::contains_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper(), gcdext_exact(), idiv_float(), input_mpq(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::intersect_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::join_restriction(), lcm_gcd_exact(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::mulmod(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator>>(), parse_number(), PPL_FUNCTION_CLASS(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::restrict(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), sqrt_mpq(), Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign(), sub_mul_int(), Parma_Polyhedra_Library::sub_restriction(), Parma_Polyhedra_Library::submod(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

10.2.4.13 From bool Type Type Parma_Polyhedra_Library::Checked::Result_Class

Definition at line 446 of file checked.defs.hh.

Referenced by input_mpq(), and PPL_FUNCTION_CLASS().

10.2.4.14 Parma_Polyhedra_Library::Checked::Result_Relation

Definition at line 430 of file checked.defs.hh.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::cmp(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::sgn().

10.2.4.15 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::To

Definition at line 450 of file checked.defs.hh.

Referenced by assign_float_float(), assign_signed_int_mpz(), assign_signed_int_signed_int(), assign_signed_int_unsigned_int(), assign_unsigned_int_mpz(), assign_unsigned_int_signed_int(), assign_unsigned_int_unsigned_int(), gcd_exact_noabs(), lcm_gcd_exact(), and PPL_FUNCTION_CLASS().

10.2.4.16 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To1

Definition at line 509 of file checked.defs.hh.

Referenced by gcdext_exact().

10.2.4.17 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To2

Definition at line 509 of file checked.defs.hh.

Referenced by gcdext_exact().

10.2.4.18 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To3

Definition at line 509 of file checked.defs.hh.

Referenced by gcdext_exact().

10.2.4.19 From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::Type

Definition at line 434 of file checked.defs.hh.

Referenced by add_2exp_float(), add_2exp_signed_int(), add_2exp_unsigned_int(), add_mul_int(), div_2exp_float(), div_2exp_signed_int(), div_2exp_unsigned_int(), div_signed_int(), div_unsigned_int(), idiv_float(), isqrtrem(), mul_2exp_float(), mul_2exp_signed_int(), mul_2exp_unsigned_int(), smod_2exp_float(), smod_2exp_signed_int(), smod_2exp_unsigned_int(), sqrt_unsigned_int(), sub_2exp_float(), sub_2exp_signed_int(), sub_2exp_unsigned_int(), sub_mul_int(), umod_2exp_float(), umod_2exp_signed_int(), and umod_2exp_unsigned_int().

10.3 Parma_Polyhedra_Library::Implementation Namespace Reference

Implementation related data and functions.

Namespaces

- namespace Octagonal_Shapes
- namespace Pointset_Powersets
- namespace Termination

Functions

- unsigned int first_one (mp_limb_t w)

  Assuming `w` *is nonzero, returns the index of the first set bit in* `w`*.*

- unsigned int last_one (mp_limb_t w)

  Assuming `w` *is nonzero, returns the index of the last set bit in* `w`*.*

Variables

- unsigned int in_assert = 0

  Non zero during evaluation of PPL_ASSERT expression.

10.3.1 Detailed Description

Implementation related data and functions.

10.3.2 Function Documentation

10.3.2.1 unsigned int Parma_Polyhedra_Library::Implementation::first_one (mp_limb_t *w*) [inline]

Assuming `w` is nonzero, returns the index of the first set bit in `w`.

Definition at line 213 of file Bit_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Bit_Row::first(), and Parma_Polyhedra_Library::Bit_Row::next().

10.3.2.2 unsigned int Parma_Polyhedra_Library::Implementation::last_one (mp_limb_t *w*) [inline]

Assuming `w` is nonzero, returns the index of the last set bit in `w`.

Definition at line 280 of file Bit_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Bit_Row::last(), and Parma_Polyhedra_Library::Bit_Row::prev().

10.3.3 Variable Documentation

10.3.3.1 unsigned int Parma_Polyhedra_Library::Implementation::in_assert = 0

Non zero during evaluation of PPL_ASSERT expression.

Definition at line 40 of file globals.cc.

Referenced by Parma_Polyhedra_Library::maybe_abandon().

10.4 Parma_Polyhedra_Library::Implementation::Octagonal_Shapes Namespace Reference

10.5 Parma_Polyhedra_Library::Implementation::Pointset_Powersets Namespace Reference

10.6 Parma_Polyhedra_Library::Implementation::Termination Namespace Reference

Functions

- void assign_all_inequalities_approximation (const Constraint_System &cs_in, Constraint_System &cs_out)
- template<typename PSET >
  void assign_all_inequalities_approximation (const PSET &pset, Constraint_System &cs)
- template<>
  void assign_all_inequalities_approximation (const C_Polyhedron &ph, Constraint_System &cs)
- void shift_unprimed_variables (Constraint_System &cs)
- template<typename PSET >
  void assign_all_inequalities_approximation (const PSET &pset_before, const PSET &pset_after, Constraint_System &cs)
- bool termination_test_MS (const Constraint_System &cs)
- bool one_affine_ranking_function_MS (const Constraint_System &cs, Generator &mu)
- void all_affine_ranking_functions_MS (const Constraint_System &cs, C_Polyhedron &mu_space)
- void all_affine_quasi_ranking_functions_MS (const Constraint_System &cs, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)
- bool termination_test_PR (const Constraint_System &cs_before, const Constraint_System &cs_after)
- bool one_affine_ranking_function_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, Generator &mu)

- void all_affine_ranking_functions_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, NNC_Polyhedron &mu_space)
- bool termination_test_PR_original (const Constraint_System &cs)
- bool one_affine_ranking_function_PR_original (const Constraint_System &cs, Generator &mu)
- void all_affine_ranking_functions_PR_original (const Constraint_System &cs, NNC_Polyhedron &mu_space)
- void fill_constraint_systems_MS (const Constraint_System &cs, Constraint_System &cs_out1, Constraint_System &cs_out2)

  Fill the constraint system(s) for the application of the Mesnard and Serebrenik improved termination tests.

- void fill_constraint_system_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, Constraint_System &cs_out, Linear_Expression &le_out)

  Fill the constraint system(s) for the application of the Podelski and Rybalchenko improved termination tests.

- void fill_constraint_system_PR_original (const Constraint_System &cs, Constraint_System &cs_out, Linear_Expression &le_out)

10.6.1 Function Documentation

10.6.1.1 void Parma_Polyhedra_Library::Implementation::Termination::all_affine_quasi_ranking_functions_MS (const Constraint_System & *cs*, C_Polyhedron & *decreasing_mu_space*, C_Polyhedron & *bounded_mu_space*)

Definition at line 558 of file termination.cc.

10.6.1.2 void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_MS (const Constraint_System & *cs*, C_Polyhedron & *mu_space*)

Definition at line 514 of file termination.cc.

10.6.1.3 void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR (const Constraint_System & *cs_before*, const Constraint_System & *cs_after*, NNC_Polyhedron & *mu_space*)

Definition at line 757 of file termination.cc.

10.6.1.4 void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR_original (const Constraint_System & *cs*, NNC_Polyhedron & *mu_space*)

Definition at line 850 of file termination.cc.

Referenced by Parma_Polyhedra_Library::all_affine_ranking_functions_PR().

10.6.1.5 template<typename PSET > void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const PSET & *pset_before*, const PSET & *pset_after*, Constraint_System & *cs*) [inline]

Definition at line 161 of file termination.templates.hh.

References assign_all_inequalities_approximation(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::insert(), and shift_unprimed_variables().

10.6.1.6 template<> void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const C_Polyhedron & *ph*, Constraint_System & *cs*) [inline]

Definition at line 58 of file termination.cc.

10.6.1.7 template<typename PSET > void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const PSET & *pset*, Constraint_System & *cs*) [inline]

Definition at line 146 of file termination.templates.hh.

References assign_all_inequalities_approximation().

10.6.1.8 void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const Constraint_System & *cs_in*, Constraint_System & *cs_out*)

Definition at line 30 of file termination.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::has_equalities(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint::is_equality(), and Parma_Polyhedra_Library::Constraint::is_strict_inequality().

Referenced by Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS(), Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS_2(), Parma_Polyhedra_Library::all_affine_ranking_functions_MS(), Parma_Polyhedra_Library::all_affine_ranking_functions_MS_2(), Parma_Polyhedra_Library::all_affine_ranking_functions_PR(), Parma_Polyhedra_Library::all_affine_ranking_functions_PR_2(), assign_all_inequalities_approximation(), Parma_Polyhedra_Library::one_affine_ranking_function_MS(), Parma_Polyhedra_Library::one_affine_ranking_function_MS_2(), Parma_Polyhedra_Library::one_affine_ranking_function_PR(), Parma_Polyhedra_Library::one_affine_ranking_function_PR_2(), Parma_Polyhedra_Library::termination_test_MS(), Parma_Polyhedra_Library::termination_test_MS_2(), Parma_Polyhedra_Library::termination_test_PR(), and Parma_Polyhedra_Library::termination_test_PR_2().

10.6.1.9 void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_system_-PR (const Constraint_System & *cs_before*, const Constraint_System & *cs_after*, Constraint_System & *cs_out*, Linear_Expression & *le_out*)

Fill the constraint system(s) for the application of the Podelski and Rybalchenko improved termination tests.

Parameters

cs_before The input constraint system describing the state *before* the execution of the loop body, where variables indices are allocated as follows:

- $x_1, \ldots, x_n$ go onto space dimensions $0, \ldots, n-1$.

The system does not contain any equality.

cs_after The input constraint system describing the state *after* the execution of the loop body, where variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$.

The system does not contain any equality.

cs_out The output constraint system, where variables indices are allocated as follows:

- u_3 goes onto space dimensions $0, \ldots, s-1$;
- u_2 goes onto space dimensions $s, \ldots, s+r-1$;
- u_1 goes onto space dimensions $s+r, \ldots, s+2r-1$.

The improved Podelski-Rybalchenko method described in the paper is based on a loop encoding of the form

$$\begin{pmatrix} A_B & \mathbf{0} \\ A_C & A'_C \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x}' \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b}_B \\ \boldsymbol{b}_C \end{pmatrix},$$

where $A_B \in \mathbb{Q}^{r \times n}$, $A_C \in \mathbb{Q}^{s \times n}$, $A'_C \in \mathbb{Q}^{s \times n}$, $\boldsymbol{b}_B \in \mathbb{Q}^r$, $\boldsymbol{b}_C \in \mathbb{Q}^s$. The corresponding system is:

$$\begin{aligned} (\boldsymbol{v}_1 - \boldsymbol{v}_2)^{\mathrm{T}} A_B - \boldsymbol{v}_3^{\mathrm{T}} A_C &= \mathbf{0}^{\mathrm{T}}, \\ \boldsymbol{v}_2^{\mathrm{T}} A_B + \boldsymbol{v}_3^{\mathrm{T}} (A_C + A'_C) &= \mathbf{0}^{\mathrm{T}}, \\ \boldsymbol{v}_2 \boldsymbol{b}_B + \boldsymbol{v}_3 \boldsymbol{b}_C &< 0, \end{aligned}$$

where $\boldsymbol{v}_1 \in \mathbb{Q}^r_+$, $\boldsymbol{v}_2 \in \mathbb{Q}^r_+$, $\boldsymbol{v}_3 \in \mathbb{Q}^s_+$. The space of ranking functions is then spanned by $\boldsymbol{v}_3^{\mathrm{T}} A'_C \boldsymbol{x}$.

In contrast, our encoding is of the form

$$\begin{pmatrix} \mathbf{0} & E_B \\ E'_C & E_C \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{x} \end{pmatrix} + \begin{pmatrix} \boldsymbol{d}_B \\ \boldsymbol{d}_C \end{pmatrix} \geq \mathbf{0},$$

where $E_B = -A_B$, $E_C = -A_C$, $E'_C = -A'_C$, $\boldsymbol{d}_B = \boldsymbol{b}_B$ and $\boldsymbol{d}_C = \boldsymbol{b}_C$. The corresponding system is:

$$\begin{aligned} (\boldsymbol{u}_1 - \boldsymbol{u}_2)^{\mathrm{T}} E_B - \boldsymbol{u}_3^{\mathrm{T}} E_C &= \mathbf{0}^{\mathrm{T}}, \\ \boldsymbol{u}_2^{\mathrm{T}} E_B + \boldsymbol{u}_3^{\mathrm{T}} (E_C + E'_C) &= \mathbf{0}^{\mathrm{T}}, \\ \boldsymbol{u}_2 \boldsymbol{d}_B + \boldsymbol{u}_3 \boldsymbol{d}_C &> 0, \end{aligned}$$

where $\boldsymbol{u}_1 \in \mathbb{Q}^r_-$, $\boldsymbol{u}_2 \in \mathbb{Q}^r_-$, $\boldsymbol{u}_3 \in \mathbb{Q}^s_-$. The space of ranking functions is then spanned by $\boldsymbol{u}_3^{\mathrm{T}} E'_C \boldsymbol{x}$.

Parameters

le_out The expression to be minimized in the context of `cs_out`: a value of -1 or less entails termination.

Definition at line 353 of file termination.cc.

10.6.1.10 void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_system_PR_original (const Constraint_System & *cs*, Constraint_System & *cs_out*, Linear_Expression & *le_out*)

Definition at line 432 of file termination.cc.

10.6.1.11 void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_systems_MS (const Constraint_System & *cs*, Constraint_System & *cs_out1*, Constraint_System & *cs_out2*)

Fill the constraint system(s) for the application of the Mesnard and Serebrenik improved termination tests.

Parameters

cs The input constraint system, where variables indices are allocated as follows:

- $x'_1, \ldots, x'_n$ go onto space dimensions $0, \ldots, n-1$,
- $x_1, \ldots, x_n$ go onto space dimensions $n, \ldots, 2n-1$.

The system does not contain any equality.

cs_out1 The first output constraint system.

cs_out2 The second output constraint system, if any: it may be an alias for `cs_out1`.

The allocation of variable indices in the output constraint systems `cs_out1` and `cs_out2` is as follows, where m is the number of constraints in `cs`:

- $\mu_1, \ldots, \mu_n$ go onto space dimensions $0, \ldots, n-1$;
- μ_0 goes onto space dimension n;
- $y_1, \ldots, y_m$ go onto space dimensions $n+1, \ldots, n+m$;

if we use the same constraint system, that is `&cs_out1 == &cs_out2`, then

- $z_1, ..., z_m, z_{m+1}, z_{m+2}$ go onto space dimensions $n+m+1, ..., n+2*m+2$;

otherwise

- $z_1, ..., z_m, z_{m+1}, z_{m+2}$ go onto space dimensions $n+1, ..., n+m+2$.

Definition at line 141 of file termination.cc.

10.6.1.12 bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_MS (const Constraint_System & *cs*, Generator & *mu*)

Definition at line 493 of file termination.cc.

10.6.1.13 bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR (const Constraint_System & *cs_before*, const Constraint_System & *cs_after*, Generator & *mu*)

Definition at line 642 of file termination.cc.

10.6.1.14 bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR_original (const Constraint_System & *cs*, Generator & *mu*)

Definition at line 703 of file termination.cc.

Referenced by Parma_Polyhedra_Library::one_affine_ranking_function_PR().

10.6.1.15 void Parma_Polyhedra_Library::Implementation::Termination::shift_unprimed_variables (Constraint_System & *cs*)

Definition at line 82 of file termination.cc.

Referenced by assign_all_inequalities_approximation().

10.6.1.16 bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_MS (const Constraint_System & *cs*)

Definition at line 484 of file termination.cc.

10.6.1.17 bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR (const Constraint_System & *cs_before*, const Constraint_System & *cs_after*)

Definition at line 611 of file termination.cc.

10.6.1.18 bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR_original (const Constraint_System & *cs*)

Definition at line 596 of file termination.cc.

Referenced by Parma_Polyhedra_Library::termination_test_PR().

10.7 Parma_Polyhedra_Library::Interfaces Namespace Reference

Data and functions related to language interfaces.

Namespaces

- namespace C

  Data and functions related to the C language interface.

- namespace Java

  Data and functions related to the Java language interface.

- namespace OCaml

  Data and functions related to the OCaml language interface.

- namespace Prolog

  Data and functions related to the Prolog language interfaces.

Functions

- bool is_necessarily_closed_for_interfaces (const Polyhedron &ph)

  Returns `true` *if and only if* `ph.topology() == NECESSARILY_CLOSED`.

10.7.1 Detailed Description

Data and functions related to language interfaces.

10.7.2 Function Documentation

10.7.2.1 bool Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces (const Polyhedron & *ph*) [inline]

Returns `true` if and only if `ph.topology() == NECESSARILY_CLOSED`.

Definition at line 404 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed().

10.8 Parma_Polyhedra_Library::Interfaces::C Namespace Reference

Data and functions related to the C language interface.

10.8.1 Detailed Description

Data and functions related to the C language interface.

10.9 Parma_Polyhedra_Library::Interfaces::Java Namespace Reference

Data and functions related to the Java language interface.

10.9.1 Detailed Description

Data and functions related to the Java language interface.

10.10 Parma_Polyhedra_Library::Interfaces::OCaml Namespace Reference

Data and functions related to the OCaml language interface.

10.10.1 Detailed Description

Data and functions related to the OCaml language interface.

10.11 Parma_Polyhedra_Library::Interfaces::Prolog Namespace Reference

Data and functions related to the Prolog language interfaces.

Namespaces

- namespace Ciao

  Data and functions related to the Ciao Prolog language interface.

- namespace GNU

  Data and functions related to the GNU Prolog language interface.

- namespace SICStus

  Data and functions related to the SICStus language interface.

- namespace SWI

  Data and functions related to the SWI-Prolog language interface.

- namespace XSB

  Data and functions related to the XSB language interface.

- namespace YAP

  Data and functions related to the YAP language interface.

10.11.1 Detailed Description

Data and functions related to the Prolog language interfaces.

10.12 Parma_Polyhedra_Library::Interfaces::Prolog::Ciao Namespace Reference

Data and functions related to the Ciao Prolog language interface.

10.12.1 Detailed Description

Data and functions related to the Ciao Prolog language interface.

10.13 Parma_Polyhedra_Library::Interfaces::Prolog::GNU Namespace Reference

Data and functions related to the GNU Prolog language interface.

10.13.1 Detailed Description

Data and functions related to the GNU Prolog language interface.

10.14 Parma_Polyhedra_Library::Interfaces::Prolog::SICStus Namespace Reference

Data and functions related to the SICStus language interface.

10.14.1 Detailed Description

Data and functions related to the SICStus language interface.

10.15 Parma_Polyhedra_Library::Interfaces::Prolog::SWI Namespace Reference

Data and functions related to the SWI-Prolog language interface.

10.15.1 Detailed Description

Data and functions related to the SWI-Prolog language interface.

10.16 Parma_Polyhedra_Library::Interfaces::Prolog::XSB Namespace Reference

Data and functions related to the XSB language interface.

10.16.1 Detailed Description

Data and functions related to the XSB language interface.

10.17 Parma_Polyhedra_Library::Interfaces::Prolog::YAP Namespace Reference

Data and functions related to the YAP language interface.

10.17.1 Detailed Description

Data and functions related to the YAP language interface.

10.18 Parma_Polyhedra_Library::Interval_NS Namespace Reference

Classes

- struct Property
- struct Scalar_As_Interval_Policy

Typedefs

- typedef Interval_Restriction_None< Interval_Info_Null< Scalar_As_Interval_Policy > > Scalar_As_Interval_Info
- typedef Interval_Restriction_None< Interval_Info_Null_Open< Scalar_As_Interval_Policy > > Scalar_As_Interval_Info_Open

Functions

- template<typename T >
  void reset_bits (T &bits)
- template<typename T >
  void reset_bit (T &bits, unsigned int bit)
- template<typename T >
  void set_bit (T &bits, unsigned int bit, bool value)
- template<typename T >
  bool get_bit (const T &bits, unsigned int bit)
- template<typename T >
  void set_bits (T &bits, unsigned int start, unsigned int len, T value)
- template<typename T >
  T get_bits (T &bits, unsigned int start, unsigned int len)
- template<typename Boundary , typename Info >
  const Boundary & f_lower (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  const Boundary & f_upper (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  const Info & f_info (const Interval< Boundary, Info > &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const T & >::type f_lower (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const T & >::type f_upper (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const Scalar_As_Interval_Info & >::type f_info (const T &)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, Scalar_As_Interval_Info_Open >::type f_info (const T &, bool open)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, bool >::type f_is_empty (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, bool >::type f_is_singleton (const T &x)

Variables

- const Property CARDINALITY_0 (Property::CARDINALITY_0_)
- const Property CARDINALITY_1 (Property::CARDINALITY_1_)
- const Property CARDINALITY_IS (Property::CARDINALITY_IS_)
- const Scalar_As_Interval_Info SCALAR_INFO

10.18.1 Typedef Documentation

10.18.1.1 typedef Interval_Restriction_None<Interval_Info_Null<Scalar_As_Interval_Policy> > Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Info

Definition at line 101 of file Interval.inlines.hh.

10.18.1.2 typedef Interval_Restriction_None<Interval_Info_Null_Open<Scalar_As_Interval_Policy> > Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Info_Open

Definition at line 105 of file Interval.inlines.hh.

10.18.2 Function Documentation

10.18.2.1 template<typename T > Enable_If<Is_Singleton<T>::value, Scalar_As_Interval_Info_Open>::type Parma_Polyhedra_Library::Interval_NS::f_info (const T &, bool *open*) [inline]

Definition at line 124 of file Interval.inlines.hh.

10.18.2.2 template<typename T > Enable_If<Is_Singleton<T>::value, const Scalar_As_Interval_Info&>::type Parma_Polyhedra_Library::Interval_NS::f_info (const T &) [inline]

Definition at line 119 of file Interval.inlines.hh.

References SCALAR_INFO.

10.18.2.3 template<typename Boundary , typename Info > const Info& Parma_Polyhedra_Library::Interval_NS::f_info (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 91 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.18.2.4 template<typename T > Enable_If<Is_Singleton<T>::value, bool>::type Parma_Polyhedra_Library::Interval_NS::f_is_empty (const T & *x*) [inline]

Definition at line 130 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::is_not_a_number().

Referenced by f_is_singleton().

10.18.2.5 template<typename T > Enable_If<Is_Singleton<T>::value, bool>::type Parma_Polyhedra_Library::Interval_NS::f_is_singleton (const T & *x*) [inline]

Definition at line 136 of file Interval.inlines.hh.

References f_is_empty().

10.18.2.6 template<typename T > Enable_If<Is_Singleton<T>::value, const T&>::type Parma_Polyhedra_Library::Interval_NS::f_lower (const T & *x*) [inline]

Definition at line 109 of file Interval.inlines.hh.

10.18.2.7 template<typename Boundary , typename Info > const Boundary& Parma_Polyhedra_Library::Interval_NS::f_lower (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 81 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::lower().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval<

Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_-assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_-Library::is_singleton_integer(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_-contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign(), and Parma_Polyhedra_-Library::sub_restriction().

10.18.2.8 template<typename T > Enable_If<Is_Singleton<T>::value, const T&>::type Parma_Polyhedra_Library::Interval_NS::f_upper (const T & *x*) [inline]

Definition at line 114 of file Interval.inlines.hh.

10.18.2.9 template<typename Boundary , typename Info > const Boundary& Parma_Polyhedra_Library::Interval_NS::f_upper (const Interval< Boundary, Info > & *x*) [inline]

Definition at line 86 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

10.18.2.10 template<typename T > bool Parma_Polyhedra_Library::Interval_NS::get_bit (const T & *bits*, unsigned int *bit*) [inline]

Definition at line 78 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_boundary_property(), Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::get_integer(), and Parma_Polyhedra_-Library::Interval_Info_Bitset< T, Policy >::get_interval_property().

10.18.2.11 template<typename T > T Parma_Polyhedra_Library::Interval_NS::get_bits (T & *bits*, unsigned int *start*, unsigned int *len*) [inline]

Definition at line 91 of file Interval_Info.defs.hh.

10.18.2.12 template<typename T > void Parma_Polyhedra_Library::Interval_NS::reset_bit (T & *bits*, unsigned int *bit*) [inline]

Definition at line 63 of file Interval_Info.defs.hh.

Referenced by set_bit().

10.18.2.13 template<typename T > void Parma_Polyhedra_Library::Interval_NS::reset_bits (T & *bits*) [inline]

Definition at line 57 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::ascii_load(), and Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::clear().

10.18.2.14 template<typename T > void Parma_Polyhedra_Library::Interval_NS::set_bit (T & *bits*, unsigned int *bit*, bool *value*) [inline]

Definition at line 69 of file Interval_Info.defs.hh.

References reset_bit().

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_boundary_property(), Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer(), and Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_interval_property().

10.18.2.15 template<typename T > void Parma_Polyhedra_Library::Interval_NS::set_bits (T & *bits*, unsigned int *start*, unsigned int *len*, T *value*) [inline]

Definition at line 84 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::ascii_load().

10.18.3 Variable Documentation

10.18.3.1 const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_0(Property::CARDINALITY_0_)

10.18.3.2 const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_1(Property::CARDINALITY_1_)

10.18.3.3 const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_IS(Property::CARDINALITY_IS_)

10.18.3.4 const Scalar_As_Interval_Info Parma_Polyhedra_Library::Interval_NS::SCALAR_INFO

Definition at line 103 of file Interval.inlines.hh.

Referenced by f_info(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

10.19 Parma_Polyhedra_Library::IO_Operators Namespace Reference

All input/output operators are confined to this namespace.

10.19.1 Detailed Description

All input/output operators are confined to this namespace. This is done so that the library's input/output operators do not interfere with those the user might want to define. In fact, it is highly unlikely that any predefined I/O operator will suit the needs of a client application. On the other hand, those applications for which the PPL I/O operator are enough can easily obtain access to them. For example, a directive like

```
using namespace Parma_Polyhedra_Library::IO_Operators;
```

would suffice for most uses. In more complex situations, such as

```
const Constraint_System& cs = ...;
copy(cs.begin(), cs.end(),
     ostream_iterator<Constraint>(cout, "\n"));
```

the Parma_Polyhedra_Library namespace must be suitably extended. This can be done as follows:

```
namespace Parma_Polyhedra_Library {
  // Import all the output operators into the main PPL namespace.
  using IO_Operators::operator<<;
}
```

10.19.2 Function Documentation

10.19.2.1 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *os*, const PIP_Tree_Node::Artificial_Parameter & *x*) [related]

Definition at line 347 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::denominator().

10.19.2.2 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *os*, const PIP_Tree_Node & *x*) [related]

Definition at line 341 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::print().

10.20 std Namespace Reference

The standard C++ namespace.

10.20.1 Detailed Description

The standard C++ namespace. The Parma Polyhedra Library conforms to the C++ standard and, in particular, as far as reserved names are concerned (17.4.3.1, [lib.reserved.names]). The PPL, however, defines several template specializations for the standard library function templates `swap()` and `iter_swap()` (25.2.2, [lib.alg.swap]), and for the class template `numeric_limits` (18.2.1, [lib.limits]).

Note

The PPL provides the specializations of the class template `numeric_limits` not only for PPL-specific numeric types, but also for the GMP types `mpz_class` and `mpq_class`. These specializations will be removed as soon as they will be provided by the C++ interface of GMP.

11 Class Documentation

11.1 Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any Struct Reference

A class that is constructible from just anything.

`#include <meta_programming.hh>`

Public Member Functions

- template<typename T >
  Any (const T &)

  The universal constructor.

11.1.1 Detailed Description

template<typename Base, typename Derived> struct Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any

A class that is constructible from just anything.

Definition at line 229 of file meta_programming.hh.

11.1.2 Constructor & Destructor Documentation

11.1.2.1 template<typename Base, typename Derived> template<typename T > Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any::Any (const T &) [inline]

The universal constructor.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.2 Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U > Class Template Reference

A template class to derive both OR_Matrix::iterator and OR_Matrix::const_iterator.

Public Types

- typedef std::random_access_iterator_tag iterator_category
- typedef Pseudo_Row< U > value_type
- typedef long difference_type
- typedef const Pseudo_Row< U > * pointer
- typedef const Pseudo_Row< U > & reference

Public Member Functions

- any_row_iterator (dimension_type n_rows)

  Constructor to build past-the-end objects.

- any_row_iterator (U &base)

  Builds an iterator pointing at the beginning of an OR_Matrix whose first element is `base`*;.*

- template<typename V >
  any_row_iterator (const any_row_iterator< V > &y)

  Copy constructor allowing the construction of a const_iterator from a non-const iterator.

- template<typename V >
  any_row_iterator & operator= (const any_row_iterator< V > &y)

  Assignment operator allowing the assignment of a non-const iterator to a const_iterator.

- reference operator* () const

  Dereference operator.

- pointer operator-> () const

  Indirect member selector.

- any_row_iterator & operator++ ()

Prefix increment operator.

- any_row_iterator operator++ (int)

  Postfix increment operator.

- any_row_iterator & operator-- ()

  Prefix decrement operator.

- any_row_iterator operator-- (int)

  Postfix decrement operator.

- reference operator[ ] (difference_type m) const

  Subscript operator.

- any_row_iterator & operator+= (difference_type m)

  Assignment-increment operator.

- any_row_iterator & operator-= (difference_type m)

  Assignment-decrement operator.

- difference_type operator- (const any_row_iterator &y) const

  Returns the difference between `*this` *and* `y`.

- any_row_iterator operator+ (difference_type m) const

  Returns the sum of `*this` *and* `m`.

- any_row_iterator operator- (difference_type m) const

  Returns the difference of `*this` *and* `m`.

- bool operator== (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is equal to* `y`.

- bool operator!= (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is different from* `y`.

- bool operator< (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is less than* `y`.

- bool operator<= (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is less than or equal to* `y`.

- bool operator> (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is greater than* `y`.

- bool operator>= (const any_row_iterator &y) const

  Returns `true` *if and only if* `*this` *is greater than or equal to* `y`.

- dimension_type row_size () const
- dimension_type index () const

Private Attributes

- Pseudo_Row< U > value

  Represents the beginning of a row.

- dimension_type e

  External index.

- dimension_type i

  Internal index: `i = (e+1)*(e+1)/2`.

Friends

- class any_row_iterator

11.2.1 Detailed Description

template<typename T>template<typename U> class Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >

A template class to derive both OR_Matrix::iterator and OR_Matrix::const_iterator.

Definition at line 202 of file OR_Matrix.defs.hh.

11.2.2 Member Typedef Documentation

11.2.2.1 template<typename T> template<typename U> typedef long Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::difference_type

Definition at line 206 of file OR_Matrix.defs.hh.

11.2.2.2 template<typename T> template<typename U> typedef std::random_access_iterator_tag Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::iterator_category

Definition at line 204 of file OR_Matrix.defs.hh.

11.2.2.3 template<typename T> template<typename U> typedef const Pseudo_Row<U>* Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::pointer

Definition at line 207 of file OR_Matrix.defs.hh.

11.2.2.4 template<typename T> template<typename U> typedef const Pseudo_Row<U>& Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::reference

Definition at line 208 of file OR_Matrix.defs.hh.

11.2.2.5 template<typename T> template<typename U> typedef Pseudo_Row<U> Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value_type

Definition at line 205 of file OR_Matrix.defs.hh.

11.2.3 Constructor & Destructor Documentation

11.2.3.1 template<typename T > template<typename U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator (dimension_type *n_rows*) [inline]

Constructor to build past-the-end objects.

Definition at line 130 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e, Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::row_size(), and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.3.2 template<typename T > template<typename U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator (U & *base*) [inline, explicit]

Builds an iterator pointing at the beginning of an OR_Matrix whose first element is `base`;.

Definition at line 144 of file OR_Matrix.inlines.hh.

11.2.3.3 template<typename T > template<typename U > template<typename V > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator (const any_row_iterator< V > & *y*) [inline]

Copy constructor allowing the construction of a const_iterator from a non-const iterator.

Definition at line 159 of file OR_Matrix.inlines.hh.

11.2.4 Member Function Documentation

11.2.4.1 template<typename T > template<typename U > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::index () const [inline]

Definition at line 347 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.2 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator!= (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is different from `y`.

Definition at line 303 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.3 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U >::reference Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator* () const [inline]

Dereference operator.

Definition at line 179 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.4 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator+ (difference_type *m*) const [inline]

Returns the sum of `*this` and `m`.

Definition at line 276 of file OR_Matrix.inlines.hh.

11.2.4.5 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator++ (int) [inline]

Postfix increment operator.

Definition at line 211 of file OR_Matrix.inlines.hh.

11.2.4.6 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > & Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator++ () [inline]

Prefix increment operator.

Definition at line 193 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e, Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::i, and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.7 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > & Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator+= (difference_type *m*) [inline]

Assignment-increment operator.

Definition at line 246 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e, Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::i, and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.8 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator- (difference_type *m*) const [inline]

Returns the difference of `*this` and `m`.

Definition at line 285 of file OR_Matrix.inlines.hh.

11.2.4.9 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U >::difference_type Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator- (const any_row_iterator< U > & *y*) const [inline]

Returns the difference between `*this` and `y`.

Definition at line 269 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.10 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator-- (int) [inline]

Postfix decrement operator.

Definition at line 237 of file OR_Matrix.inlines.hh.

11.2.4.11 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > & Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator-- () [inline]

Prefix decrement operator.

Definition at line 220 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e, Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::i, and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.12 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U > & Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator-= (difference_type *m*) [inline]

Assignment-decrement operator.

Definition at line 262 of file OR_Matrix.inlines.hh.

11.2.4.13 template<typename T > template<typename U > OR_Matrix< T >::template any_row_iterator< U >::pointer Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator-> () const [inline]

Indirect member selector.

Definition at line 186 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.14 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator< (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is less than `y`.

Definition at line 310 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.15 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator<= (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is less than or equal to `y`.

Definition at line 318 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.16 template<typename T > template<typename U > template<typename V > OR_Matrix< T >::template any_row_iterator< U > & Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator= (const any_row_iterator< V > & *y*) [inline]

Assignment operator allowing the assignment of a non-const iterator to a const_iterator.

Definition at line 169 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e, Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::i, and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value.

11.2.4.17 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator== (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is equal to `y`.

Definition at line 295 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.18 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator> (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is greater than `y`.

Definition at line 325 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.19 template<typename T > template<typename U > bool Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator>= (const any_row_iterator< U > & *y*) const [inline]

Returns `true` if and only if `*this` is greater than or equal to `y`.

Definition at line 333 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

11.2.4.20 template<typename T> template<typename U> reference Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator[ ] (difference_type *m*) const

Subscript operator.

11.2.4.21 template<typename T > template<typename U > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::row_size () const [inline]

Definition at line 340 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator().

11.2.5 Friends And Related Function Documentation

11.2.5.1 template<typename T> template<typename U> friend class any_row_iterator [friend]

Definition at line 317 of file OR_Matrix.defs.hh.

11.2.6 Member Data Documentation

11.2.6.1 template<typename T> template<typename U> dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::e [private]

External index.

Definition at line 305 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::index(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator!=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator++(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator+=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator-(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator--(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator<(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator<=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator==(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator>(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator>=(), and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::row_size().

11.2.6.2 template<typename T> template<typename U> dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::i [private]

Internal index: `i = (e+1)*(e+1)/2`.

Definition at line 308 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator++(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator+=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator--(), and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator=().

11.2.6.3 template<typename T> template<typename U> Pseudo_Row<U> Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::value [private]

Represents the beginning of a row.

Definition at line 302 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::any_row_iterator(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator*(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator++(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator+=(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator--(), Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator->(), and Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >::operator=().

The documentation for this class was generated from the following files:

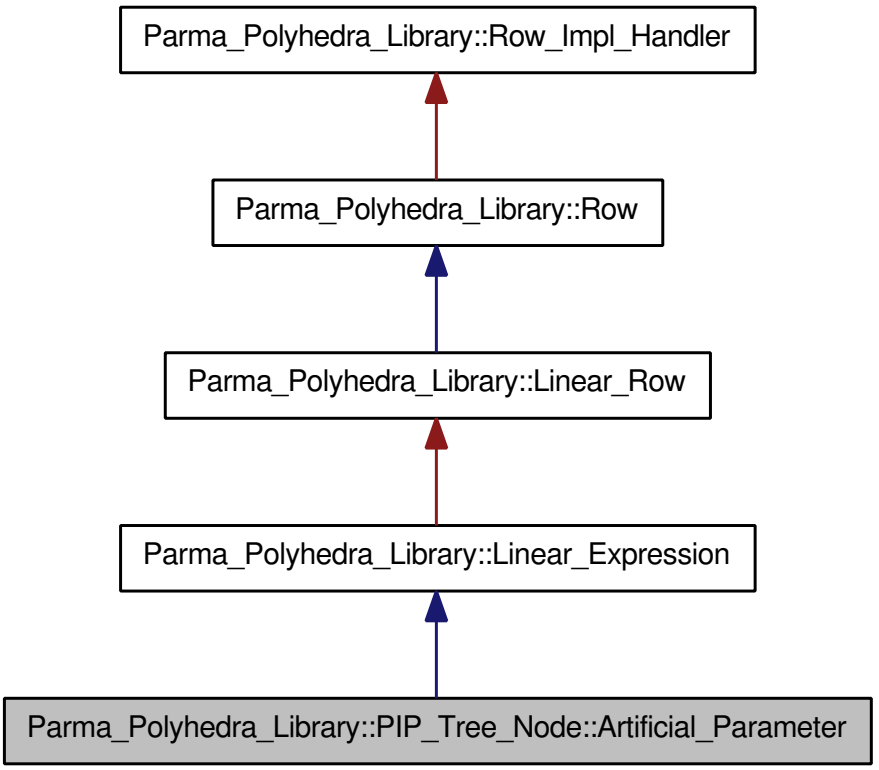
- OR_Matrix.defs.hh
- OR_Matrix.inlines.hh

11.3 Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter Class Reference

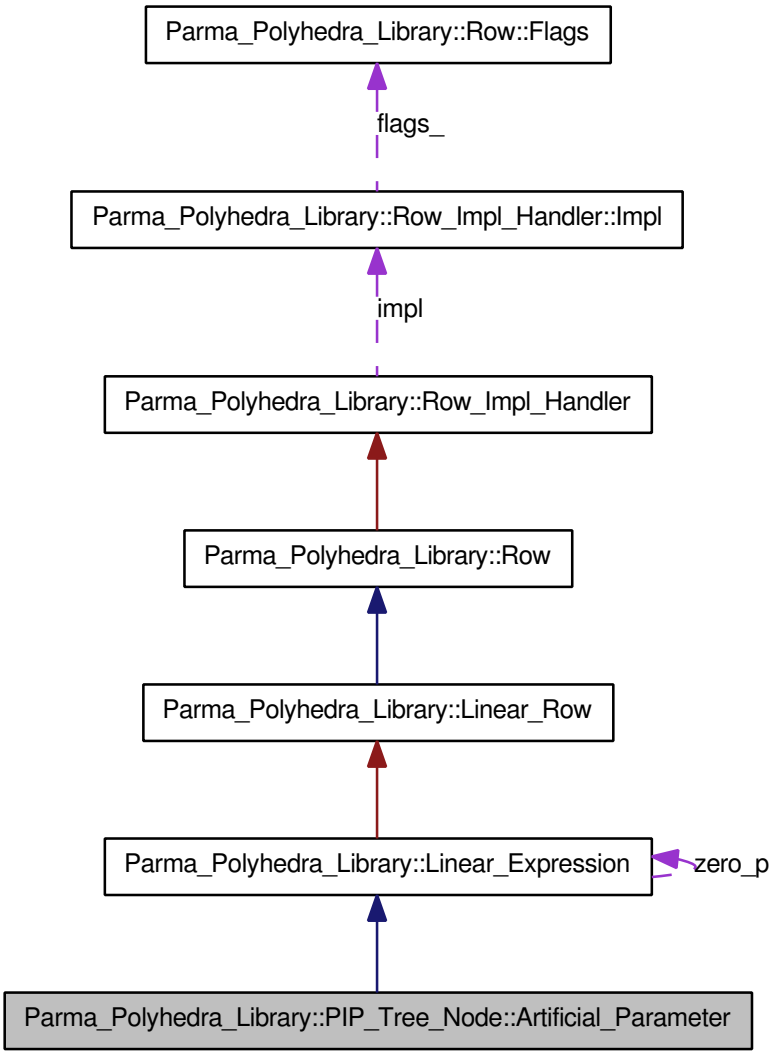
Artificial parameters in PIP solution trees.

`#include <PIP_Tree.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter:



Collaboration diagram for Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter:



Public Member Functions

- Artificial_Parameter ()

  Default constructor: builds a zero artificial parameter.

- Artificial_Parameter (const Linear_Expression &expr, Coefficient_traits::const_reference den)

  Constructor.

- Artificial_Parameter (const Artificial_Parameter &y)

  Copy constructor.

- Coefficient_traits::const_reference denominator () const

  Returns the normalized (i.e., positive) denominator.

- void swap (Artificial_Parameter &y)

Swaps `*this` *with* `y`.

- bool operator== (const Artificial_Parameter &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are equal.*

- bool operator!= (const Artificial_Parameter &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Returns `true` *if and only if the parameter is well-formed.*

Private Attributes

- Coefficient denom

  The normalized (i.e., positive) denominator.

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &os, const PIP_Tree_Node::Artificial_Parameter &x)

  Output operator.

11.3.1 Detailed Description

Artificial parameters in PIP solution trees. These parameters are built from a linear expression combining other parameters (constant term included) divided by a positive integer denominator. Coefficients at variables indices corresponding to PIP problem variables are always zero.

Definition at line 276 of file PIP_Tree.defs.hh.

11.3.2 Constructor & Destructor Documentation

11.3.2.1 Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter () [inline]

Default constructor: builds a zero artificial parameter.

Definition at line 107 of file PIP_Tree.inlines.hh.

References OK().

11.3.2.2 Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter (const Linear_Expression & *expr*, Coefficient_traits::const_reference *den*)

Constructor.

Builds artificial parameter $\frac{expr}{den}$.

Parameters

expr The expression that, after normalization, will form the numerator of the artificial parameter.

den The integer constant thatm after normalization, will form the denominator of the artificial parameter.

Exceptions

std::invalid_argument Thrown if `den` is zero.

Normalization will ensure that the denominator is positive.

Definition at line 370 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Linear_Expression::add_mul_assign, Parma_Polyhedra_Library::Linear_Expression::coefficient(), denom, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::neg_assign(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.3.2.3 Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter (const Artificial_Parameter & *y*) [inline]

Copy constructor.

Definition at line 114 of file PIP_Tree.inlines.hh.

References OK().

11.3.3 Member Function Documentation

11.3.3.1 void Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Definition at line 458 of file PIP_Tree.cc.

References ascii_dump(), and denom.

11.3.3.2 void Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Referenced by ascii_dump().

11.3.3.3 bool Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Definition at line 465 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Linear_Expression::ascii_load(), denom, and OK().

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::ascii_load().

11.3.3.4 Coefficient_traits::const_reference Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::denominator () const [inline]

Returns the normalized (i.e., positive) denominator.

Definition at line 120 of file PIP_Tree.inlines.hh.

References denom.

Referenced by Parma_Polyhedra_Library::IO_Operators::operator<<().

11.3.3.5 memory_size_type Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Definition at line 2839 of file PIP_Tree.cc.

References denom.

Referenced by total_memory_in_bytes().

11.3.3.6 bool Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::OK () const

Returns `true` if and only if the parameter is well-formed.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Definition at line 446 of file PIP_Tree.cc.

References denom.

Referenced by Artificial_Parameter(), and ascii_load().

11.3.3.7 bool Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::operator!= (const Artificial_Parameter & *y*) const

Returns `true` if and only if `*this` and `y` are different.

Definition at line 441 of file PIP_Tree.cc.

References operator==().

11.3.3.8 bool Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::operator== (const Artificial_Parameter & *y*) const

Returns `true` if and only if `*this` and `y` are equal.

Note that two artificial parameters having different space dimensions are considered to be different.

Definition at line 425 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), denom, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by operator!=().

11.3.3.9 void Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

11.3.3.10 void Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::swap (Artificial_Parameter & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 125 of file PIP_Tree.inlines.hh.

References denom.

11.3.3.11 memory_size_type Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::total_memory_in_bytes () const

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Expression.

Definition at line 2845 of file PIP_Tree.cc.

References external_memory_in_bytes().

11.3.4 Friends And Related Function Documentation

11.3.4.1 std::ostream & operator<< (std::ostream & *os*, const PIP_Tree_Node::Artificial_Parameter & *x*) [related]

Output operator.

Definition at line 347 of file PIP_Tree.cc.

11.3.5 Member Data Documentation

11.3.5.1 Coefficient Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::denom [private]

The normalized (i.e., positive) denominator.

Definition at line 339 of file PIP_Tree.defs.hh.

Referenced by Artificial_Parameter(), ascii_dump(), ascii_load(), denominator(), external_memory_in_bytes(), OK(), operator==(), and swap().

The documentation for this class was generated from the following files:

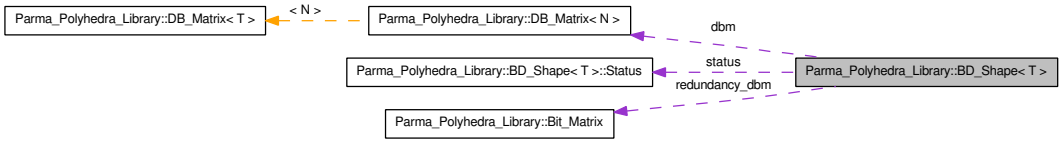
- PIP_Tree.defs.hh
- PIP_Tree.inlines.hh
- PIP_Tree.cc

11.4 Parma_Polyhedra_Library::BD_Shape< T > Class Template Reference

A bounded difference shape.

`#include <BD_Shape.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::BD_Shape< T >:



Classes

- class Status

  A conjunctive assertion about a BD_Shape<T> object.

Public Types

- typedef T coefficient_type_base

  The numeric base type upon which bounded differences are built.

- typedef N coefficient_type

  The (extended) numeric type of the inhomogeneous term of the inequalities defining a BDS.

Public Member Functions

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- int32_t hash_code () const

  Returns a 32-bit hash code for `*this`.

Constructors, Assignment, Swap and Destructor

- BD_Shape (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)
  Builds a universe or empty BDS of the specified space dimension.
- BD_Shape (const BD_Shape &y, Complexity_Class complexity=ANY_COMPLEXITY)
  Ordinary copy constructor.
- template<typename U >
  BD_Shape (const BD_Shape< U > &y, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a conservative, upward approximation of `y`.
- BD_Shape (const Constraint_System &cs)
  Builds a BDS from the system of constraints `cs`.
- BD_Shape (const Congruence_System &cgs)
  Builds a BDS from a system of congruences.
- BD_Shape (const Generator_System &gs)
  Builds a BDS from the system of generators `gs`.
- BD_Shape (const Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a BDS from the polyhedron `ph`.
- template<typename Interval >
  BD_Shape (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a BDS out of a box.
- BD_Shape (const Grid &grid, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a BDS out of a grid.
- template<typename U >
  BD_Shape (const Octagonal_Shape< U > &os, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a BDS from an octagonal shape.
- BD_Shape & operator= (const BD_Shape &y)
  The assignment operator (`*this` *and* `y` *can be dimension-incompatible).*
- void swap (BD_Shape &y)
  Swaps `*this` *with* `y` *(*`*this` *and* `y` *can be dimension-incompatible).*
- ~BD_Shape ()
  Destructor.

Member Functions that Do Not Modify the BD_Shape

- dimension_type space_dimension () const
  Returns the dimension of the vector space enclosing `*this`.
- dimension_type affine_dimension () const
  Returns 0, *if* `*this` *is empty; otherwise, returns the affine dimension of* `*this`.

- Constraint_System constraints () const
  Returns a system of constraints defining `*this`.

- Constraint_System minimized_constraints () const
  Returns a minimized system of constraints defining `*this`.

- Congruence_System congruences () const
  Returns a system of (equality) congruences satisfied by `*this`.

- Congruence_System minimized_congruences () const
  Returns a minimal system of (equality) congruences satisfied by `*this` *with the same affine dimension as* `*this`.

- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from above in* `*this`.

- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from below in* `*this`.

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool frequency (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const
  Returns `true` *if and only if there exist a unique value* `val` *such that* `*this` *saturates the equality* `expr = val`.

- bool contains (const BD_Shape &y) const
  Returns `true` *if and only if* `*this` *contains* `y`.

- bool strictly_contains (const BD_Shape &y) const
  Returns `true` *if and only if* `*this` *strictly contains* `y`.

- bool is_disjoint_from (const BD_Shape &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*

- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between `*this` *and the constraint* `c`.

- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between `*this` *and the congruence* `cg`.
- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between `*this` *and the generator* `g`.
- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty BDS.*
- bool is_universe () const
  Returns `true` *if and only if* `*this` *is a universe BDS.*
- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*
- bool is_topologically_closed () const
  Returns `true` *if and only if* `*this` *is a topologically closed subset of the vector space.*
- bool is_bounded () const
  Returns `true` *if and only if* `*this` *is a bounded BDS.*
- bool contains_integer_point () const
  Returns `true` *if and only if* `*this` *contains at least one integer point.*
- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.
- bool OK () const
  Returns `true` *if and only if* `*this` *satisfies all its invariants.*

Space-Dimension Preserving Member Functions that May Modify the BD_Shape

- void add_constraint (const Constraint &c)
  Adds a copy of constraint `c` *to the system of bounded differences defining* `*this`.
- void add_congruence (const Congruence &cg)
  Adds a copy of congruence `cg` *to the system of congruences of* `*this`.
- void add_constraints (const Constraint_System &cs)
  Adds the constraints in `cs` *to the system of bounded differences defining* `*this`.
- void add_recycled_constraints (Constraint_System &cs)
  Adds the constraints in `cs` *to the system of constraints of* `*this`.
- void add_congruences (const Congruence_System &cgs)
  Adds to `*this` *constraints equivalent to the congruences in* `cgs`.
- void add_recycled_congruences (Congruence_System &cgs)
  Adds to `*this` *constraints equivalent to the congruences in* `cgs`.
- void refine_with_constraint (const Constraint &c)
  Uses a copy of constraint `c` *to refine the system of bounded differences defining* `*this`.
- void refine_with_congruence (const Congruence &cg)

Uses a copy of congruence `cg` *to refine the system of bounded differences of* `*this`.

- void refine_with_constraints (const Constraint_System &cs)
  Uses a copy of the constraints in `cs` *to refine the system of bounded differences defining* `*this`.

- void refine_with_congruences (const Congruence_System &cgs)
  Uses a copy of the congruences in `cgs` *to refine the system of bounded differences defining* `*this`.

- void unconstrain (Variable var)
  Computes the cylindrification of `*this` *with respect to space dimension* `var`*, assigning the result to* `*this`.

- void unconstrain (const Variables_Set &vars)
  Computes the cylindrification of `*this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `*this`.

- void intersection_assign (const BD_Shape &y)
  Assigns to `*this` *the intersection of* `*this` *and* `y`.

- void upper_bound_assign (const BD_Shape &y)
  Assigns to `*this` *the smallest BDS containing the union of* `*this` *and* `y`.

- bool upper_bound_assign_if_exact (const BD_Shape &y)
  If the upper bound of `*this` *and* `y` *is exact, it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- bool integer_upper_bound_assign_if_exact (const BD_Shape &y)
  If the integer upper bound of `*this` *and* `y` *is exact, it is assigned to* `*this` *and* `true` *is returned; otherwise* `false` *is returned.*

- void difference_assign (const BD_Shape &y)
  Assigns to `*this` *the smallest BD shape containing the set difference of* `*this` *and* `y`.

- bool simplify_using_context_assign (const BD_Shape &y)
  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`*. If* `false` *is returned, then the intersection is empty.*

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the affine image of* `*this` *under the function mapping variable* `var` *into the affine expression specified by* `expr` *and* `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *into the affine expression specified by* `expr` *and* `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the image of* `*this` *with respect to the affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)
  Assigns to `*this` *the image of* `*this` *with respect to the affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the preimage of* `*this` *with respect to the affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void time_elapse_assign (const BD_Shape &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`.

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)

  Wraps the specified dimensions of the vector space.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates for the space dimensions corresponding to* `vars`.

- void topological_closure_assign ()

  Assigns to `*this` *its topological closure.*

- void CC76_extrapolation_assign (const BD_Shape &y, unsigned *tp=0)

  Assigns to `*this` *the result of computing the CC76-extrapolation between* `*this` *and* `y`.

- template<typename Iterator >
  void CC76_extrapolation_assign (const BD_Shape &y, Iterator first, Iterator last, unsigned *tp=0)

  Assigns to `*this` *the result of computing the CC76-extrapolation between* `*this` *and* `y`.

- void BHMZ05_widening_assign (const BD_Shape &y, unsigned *tp=0)

  Assigns to `*this` *the result of computing the BHMZ05-widening of* `*this` *and* `y`.

- void limited_BHMZ05_extrapolation_assign (const BD_Shape &y, const Constraint_System &cs, unsigned *tp=0)

  Improves the result of the BHMZ05-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `*this`.

- void CC76_narrowing_assign (const BD_Shape &y)
  Assigns to `*this` the result of restoring in `y` the constraints of `*this` that were lost by CC76-extrapolation applications.

- void limited_CC76_extrapolation_assign (const BD_Shape &y, const Constraint_System &cs, unsigned *tp=0)
  Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

- void H79_widening_assign (const BD_Shape &y, unsigned *tp=0)
  Assigns to `*this` the result of computing the H79-widening between `*this` and `y`.

- void widening_assign (const BD_Shape &y, unsigned *tp=0)
  Same as H79_widening_assign(y, tp).

- void limited_H79_extrapolation_assign (const BD_Shape &y, const Constraint_System &cs, unsigned *tp=0)
  Improves the result of the H79-widening computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)
  Adds `m` new dimensions and embeds the old BDS into the new space.

- void add_space_dimensions_and_project (dimension_type m)
  Adds `m` new dimensions to the BDS and does not embed it in the new vector space.

- void concatenate_assign (const BD_Shape &y)
  Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

- void remove_space_dimensions (const Variables_Set &vars)
  Removes all the specified dimensions.

- void remove_higher_space_dimensions (dimension_type new_dimension)
  Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)
  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)
  Creates `m` copies of the space dimension corresponding to `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` into `dest`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension that a BDS can handle.

- static bool can_recycle_constraint_systems ()

  Returns `false` *indicating that this domain cannot recycle constraints.*

- static bool can_recycle_congruence_systems ()

  Returns `false` *indicating that this domain cannot recycle congruences.*

Private Types

- typedef Checked_Number< T, Debug_WRD_Extended_Number_Policy > N

  The (extended) numeric type of the inhomogeneous term of the inequalities defining a BDS.

Private Member Functions

- bool marked_zero_dim_univ () const

  Returns `true` *if the BDS is the zero-dimensional universe.*

- bool marked_empty () const

  Returns `true` *if the BDS is known to be empty.*

- bool marked_shortest_path_closed () const

  Returns `true` *if the system of bounded differences is known to be shortest-path closed.*

- bool marked_shortest_path_reduced () const

  Returns `true` *if the system of bounded differences is known to be shortest-path reduced.*

- void set_empty ()

  Turns `*this` *into an empty BDS.*

- void set_zero_dim_univ ()

  Turns `*this` *into an zero-dimensional universe BDS.*

- void set_shortest_path_closed ()

  Marks `*this` *as shortest-path closed.*

- void set_shortest_path_reduced ()

  Marks `*this` *as shortest-path closed.*

- void reset_shortest_path_closed ()

  Marks `*this` *as possibly not shortest-path closed.*

- void reset_shortest_path_reduced ()

  Marks `*this` *as possibly not shortest-path reduced.*

- void shortest_path_closure_assign () const

  Assigns to `this->dbm` *its shortest-path closure.*

- void shortest_path_reduction_assign () const

  Assigns to `this->dbm` *its shortest-path closure and records into* `this->redundancy_dbm` *which of the entries in* `this->dbm` *are redundant.*

- bool is_shortest_path_reduced () const

  Returns `true` *if and only if* `this->dbm` *is shortest-path closed and* `this->redundancy_dbm` *correctly flags the redundant entries in* `this->dbm`*.*

- void incremental_shortest_path_closure_assign (Variable var) const

  Incrementally computes shortest-path closure, assuming that only constraints affecting variable `var` *need to be considered.*

- bool bounds (const Linear_Expression &expr, bool from_above) const

  Checks if and how `expr` *is bounded in* `*this`*.*

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included, Generator &g) const

  Maximizes or minimizes `expr` *subject to* `*this`*.*

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included) const

  Maximizes or minimizes `expr` *subject to* `*this`*.*

- bool BFT00_upper_bound_assign_if_exact (const BD_Shape &y)

  If the upper bound of `*this` *and* `y` *is exact it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- template<bool integer_upper_bound>
  bool BHZ09_upper_bound_assign_if_exact (const BD_Shape &y)

  If the upper bound of `*this` *and* `y` *is exact it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- void refine_no_check (const Constraint &c)

  Uses the constraint `c` *to refine* `*this`*.*

- void refine_no_check (const Congruence &cg)

  Uses the congruence `cg` *to refine* `*this`*.*

- void add_dbm_constraint (dimension_type i, dimension_type j, const N &k)

  Adds the constraint `dbm[i][j] <= k`*.*

- void add_dbm_constraint (dimension_type i, dimension_type j, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den)

  Adds the constraint `dbm[i][j] <= num/den`*.*

- void refine (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Adds to the BDS the constraint var $\bowtie \frac{\text{expr}}{\text{denominator}}$.

- void forget_all_dbm_constraints (dimension_type v)

  Removes all the constraints on row/column `v`*.*

- void forget_binary_dbm_constraints (dimension_type v)

  Removes all binary constraints on row/column `v`.

- void deduce_v_minus_u_bounds (dimension_type v, dimension_type last_v, const Linear_Expression &sc_expr, Coefficient_traits::const_reference sc_den, const N &ub_v)

  An helper function for the computation of affine relations.

- void deduce_u_minus_v_bounds (dimension_type v, dimension_type last_v, const Linear_Expression &sc_expr, Coefficient_traits::const_reference sc_den, const N &minus_lb_v)

  An helper function for the computation of affine relations.

- void get_limiting_shape (const Constraint_System &cs, BD_Shape &limiting_shape) const

  Adds to `limiting_shape` *the bounded differences in* `cs` *that are satisfied by* `*this`.

- void compute_predecessors (std::vector< dimension_type > &predecessor) const

  Compute the (zero-equivalence classes) predecessor relation.

- void compute_leaders (std::vector< dimension_type > &leaders) const

  Compute the leaders of zero-equivalence classes.

- void drop_some_non_integer_points_helper (N &elem)

Private Attributes

- DB_Matrix< N > dbm

  The matrix representing the system of bounded differences.

- Status status

  The status flags to keep track of the internal state.

- Bit_Matrix redundancy_dbm

  A matrix indicating which constraints are redundant.

Friends

- class Parma_Polyhedra_Library::BD_Shape
- class Parma_Polyhedra_Library::Box
- bool operator== (const BD_Shape< T > &x, const BD_Shape< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same BDS.*

- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< U > &x, const BD_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)
- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< U > &x, const BD_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< U > &x, const BD_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)
- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const BD_Shape< T > &c)

Related Functions

(Note that these are not member functions.)

- template<typename T >
  std::ostream & operator<< (std::ostream &s, const BD_Shape< T > &bds)

  Output operator.

- template<typename T >
  bool operator!= (const BD_Shape< T > &x, const BD_Shape< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *aren't the same BDS.*

- template<typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir)

  Computes the L_∞ *distance between* `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const BD_Shape< T > &x, const BD_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ *distance between* `x` *and* `y`*.*

- bool extract_bounded_difference (const Constraint &c, dimension_type c_space_dim, dimension_type &c_num_vars, dimension_type &c_first_var, dimension_type &c_second_var, Coefficient &c_coeff)

  Decodes the constraint `c` *as a bounded difference.*

- void compute_leader_indices (const std::vector< dimension_type > &predecessor, std::vector< dimension_type > &indices)

  Extracts leader indices from the predecessor relation.

- template<typename T >
  void swap (Parma_Polyhedra_Library::BD_Shape< T > &x, Parma_Polyhedra_Library::BD_Shape< T > &y)

  Specializes `std::swap`.

Exception Throwers

- void throw_dimension_incompatible (const char *method, const BD_Shape &x) const
- void throw_dimension_incompatible (const char *method, dimension_type required_dim) const
- void throw_dimension_incompatible (const char *method, const Constraint &c) const
- void throw_dimension_incompatible (const char *method, const Congruence &cg) const
- void throw_dimension_incompatible (const char *method, const Generator &g) const
- void throw_dimension_incompatible (const char *method, const char *name_row, const Linear_Expression &y) const
- static void throw_expression_too_complex (const char *method, const Linear_Expression &e)
- static void throw_generic (const char *method, const char *reason)

11.4.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::BD_Shape< T >

A bounded difference shape. The class template BD_Shape<T> allows for the efficient representation of a restricted kind of *topologically closed* convex polyhedra called *bounded difference shapes* (BDSs, for short). The name comes from the fact that the closed affine half-spaces that characterize the polyhedron can be expressed by constraints of the form $\pm x_i \leq k$ or $x_i - x_j \leq k$, where the inhomogeneous term k is a rational number.

Based on the class template type parameter `T`, a family of extended numbers is built and used to approximate the inhomogeneous term of bounded differences. These extended numbers provide a representation for the value $+\infty$, as well as *rounding-aware* implementations for several arithmetic functions. The value of the type parameter `T` may be one of the following:

- a bounded precision integer type (e.g., `int32_t` or `int64_t`);
- a bounded precision floating point type (e.g., `float` or `double`);
- an unbounded integer or rational type, as provided by GMP (i.e., `mpz_class` or `mpq_class`).

The user interface for BDSs is meant to be as similar as possible to the one developed for the polyhedron class C_Polyhedron.

The domain of BD shapes *optimally supports*:

- tautological and inconsistent constraints and congruences;
- bounded difference constraints;
- non-proper congruences (i.e., equalities) that are expressible as bounded-difference constraints.

Depending on the method, using a constraint or congruence that is not optimally supported by the domain will either raise an exception or result in a (possibly non-optimal) upward approximation.

A constraint is a bounded difference if it has the form

$$a_i x_i - a_j x_j \bowtie b$$

where $\bowtie \in \{\leq, =, \geq\}$ and a_i, a_j, b are integer coefficients such that $a_i = 0$, or $a_j = 0$, or $a_i = a_j$. The user is warned that the above bounded difference Constraint object will be mapped into a *correct* and *optimal* approximation that, depending on the expressive power of the chosen template argument `T`, may loose some precision. Also note that strict constraints are not bounded differences.

For instance, a Constraint object encoding $3x - 3y \leq 1$ will be approximated by:

- $x - y \leq 1$, if `T` is a (bounded or unbounded) integer type;
- $x - y \leq \frac{1}{3}$, if `T` is the unbounded rational type `mpq_class`;
- $x - y \leq k$, where $k > \frac{1}{3}$, if `T` is a floating point type (having no exact representation for $\frac{1}{3}$).

On the other hand, depending from the context, a Constraint object encoding $3x - y \leq 1$ will be either upward approximated (e.g., by safely ignoring it) or it will cause an exception.

In the following examples it is assumed that the type argument `T` is one of the possible instances listed above and that variables `x`, `y` and `z` are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds a BDS corresponding to a cube in $\mathbb{R}^3$, given as a system of constraints:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 1);
cs.insert(y >= 0);
cs.insert(y <= 1);
cs.insert(z >= 0);
cs.insert(z <= 1);
BD_Shape<T> bd(cs);
```

Since only those constraints having the syntactic form of a *bounded difference* are optimally supported, the following code will throw an exception (caused by constraints 7, 8 and 9):

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 1);
cs.insert(y >= 0);
cs.insert(y <= 1);
cs.insert(z >= 0);
cs.insert(z <= 1);
cs.insert(x + y <= 0);       // 7
cs.insert(x - z + x >= 0);   // 8
cs.insert(3*z - y <= 1);     // 9
BD_Shape<T> bd(cs);
```

Definition at line 403 of file BD_Shape.defs.hh.

11.4.2 Member Typedef Documentation

11.4.2.1 template<typename T> typedef N Parma_Polyhedra_Library::BD_Shape< T >::coefficient_type

The (extended) numeric type of the inhomogeneous term of the inequalities defining a BDS.

Definition at line 423 of file BD_Shape.defs.hh.

11.4.2.2 template<typename T> typedef T Parma_Polyhedra_Library::BD_Shape< T >::coefficient_type_base

The numeric base type upon which bounded differences are built.

Definition at line 417 of file BD_Shape.defs.hh.

11.4.2.3 template<typename T> typedef Checked_Number<T, Debug_WRD_Extended_Number_Policy> Parma_Polyhedra_Library::BD_Shape< T >::N [private]

The (extended) numeric type of the inhomogeneous term of the inequalities defining a BDS.

Definition at line 410 of file BD_Shape.defs.hh.

11.4.3 Constructor & Destructor Documentation

11.4.3.1 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds a universe or empty BDS of the specified space dimension.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the BDS;

kind Specifies whether the universe or the empty BDS has to be built.

Definition at line 115 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed().

11.4.3.2 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const BD_Shape< T > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 130 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), and Parma_-Polyhedra_Library::BD_Shape< T >::redundancy_dbm.

11.4.3.3 template<typename T > template<typename U > Parma_Polyhedra_Library::BD_-Shape< T >::BD_Shape (const BD_Shape< U > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a conservative, upward approximation of `y`.

The complexity argument is ignored.

Definition at line 139 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_-Library::BD_Shape< T >::marked_zero_dim_univ(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ().

11.4.3.4 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Constraint_System & *cs*) [inline, explicit]

Builds a BDS from the system of constraints `cs`.

The BDS inherits the space dimension of `cs`.

Parameters

cs A system of BD constraints.

Exceptions

std::invalid_argument Thrown if `cs` contains a constraint which is not optimally supported by the BD shape domain.

Definition at line 269 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_constraints(), Parma_Polyhedra_-Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::Constraint_-System::space_dimension().

11.4.3.5 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Congruence_System & *cgs*) [inline, explicit]

Builds a BDS from a system of congruences.

The BDS inherits the space dimension of `cgs`

Parameters

cgs A system of congruences.

Exceptions

std::invalid_argument Thrown if `cgs` contains congruences which are not optimally supported by the BD shape domain.

Definition at line 48 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_congruences().

11.4.3.6 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Generator_System & *gs*) [inline, explicit]

Builds a BDS from the system of generators `gs`.

Builds the smallest BDS containing the polyhedron defined by `gs`. The BDS inherits the space dimension of `gs`.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 56 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_generic(), and Parma_Polyhedra_Library::Generator::type().

11.4.3.7 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a BDS from the polyhedron `ph`.

Builds a BDS containing `ph` using algorithms whose complexity does not exceed the one specified by `complexity`. If `complexity` is `ANY_COMPLEXITY`, then the BDS built is the smallest one containing `ph`.

Definition at line 174 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::ANY_-

COMPLEXITY, Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Polyhedron::con_sys, Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Polyhedron::constraints_are_minimized(), Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date(), Parma_Polyhedra_Library::Polyhedron::has_pending_constraints(), Parma_Polyhedra_Library::Polyhedron::has_something_pending(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::MIP_Problem::is_satisfiable(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Polyhedron::is_universe(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, Parma_Polyhedra_Library::MIP_Problem::optimizing_point(), Parma_Polyhedra_Library::POLYNOMIAL_COMPLEXITY, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::MIP_Problem::set_objective_function(), Parma_Polyhedra_Library::MIP_Problem::set_optimization_mode(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), Parma_Polyhedra_Library::SIMPLEX_COMPLEXITY, Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Polyhedron::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.4.3.8 template<typename T > template<typename Interval > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a BDS out of a box.

The BDS inherits the space dimension of the box. The built BDS is the most precise BDS that includes the box.

Parameters

box The box representing the BDS to be built.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 280 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.4.3.9 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Grid & *grid*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a BDS out of a grid.

The BDS inherits the space dimension of the grid. The built BDS is the most precise BDS that includes the grid.

Parameters

grid The grid used to build the BDS.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `grid` exceeds the maximum allowed space dimension.

Definition at line 295 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Grid::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::Grid::space_dimension().

11.4.3.10 template<typename T > template<typename U > Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape (const Octagonal_Shape< U > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a BDS from an octagonal shape.

The BDS inherits the space dimension of the octagonal shape. The built BDS is the most precise BDS that includes the octagonal shape.

Parameters

os The octagonal shape used to build the BDS.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `os` exceeds the maximum allowed space dimension.

Definition at line 308 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

11.4.3.11 template<typename T > Parma_Polyhedra_Library::BD_Shape< T >::~BD_Shape () [inline]

Destructor.

Definition at line 336 of file BD_Shape.inlines.hh.

11.4.4 Member Function Documentation

11.4.4.1 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_congruence (const Congruence & *cg*) [inline]

Adds a copy of congruence `cg` to the system of congruences of `*this`.

Parameters

cg The congruence to be added.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible, or `cg` is not optimally supported by the BD shape domain.

Definition at line 447 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruences().

11.4.4.2 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_congruences (const Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs Contains the congruences that will be added to the system of constraints of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the BD shape domain.

Definition at line 174 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Congruence_System::begin(), and Parma_Polyhedra_Library::Congruence_System::end().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_recycled_congruences(), and Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape().

11.4.4.3 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_constraint (const Constraint & *c*) [inline]

Adds a copy of constraint `c` to the system of bounded differences defining `*this`.

Parameters

c The constraint to be added.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible, or `c` is not optimally supported by the BD shape domain.

Definition at line 374 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::difference_assign().

11.4.4.4 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_constraints (const Constraint_System & *cs*) [inline]

Adds the constraints in `cs` to the system of bounded differences defining `*this`.

Parameters

cs The constraints that will be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible, or `cs` contains a constraint which is not optimally supported by the BD shape domain.

Definition at line 160 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), and Parma_Polyhedra_Library::Constraint_System::end().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_recycled_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape().

11.4.4.5 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint (dimension_type *i*, dimension_type *j*, Coefficient_traits::const_reference *num*, Coefficient_traits::const_reference *den*) [inline, private]

Adds the constraint `dbm[i][j] <= num/den`.

Definition at line 672 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::div_round_up(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.6 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint (dimension_type *i*, dimension_type *j*, const N & *k*) [inline, private]

Adds the constraint `dbm[i][j] <= k`.

Definition at line 657 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::refine().

11.4.4.7 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_recycled_congruences (Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs Contains the congruences that will be added to the system of constraints of `*this`. Its elements may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the BD shape domain.

Warning

The only assumption that can be made on `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 182 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_congruences().

11.4.4.8 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_recycled_constraints (Constraint_System & *cs*) [inline]

Adds the constraints in `cs` to the system of constraints of `*this`.

Parameters

cs The constraint system to be added to `*this`. The constraints in `cs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible, or `cs` contains a constraint which is not optimally supported by the BD shape domain.

Warning

The only assumption that can be made on `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 168 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_constraints().

11.4.4.9 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed (dimension_type *m*) [inline]

Adds `m` new dimensions and embeds the old BDS into the new space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new BDS, which is defined by a system of bounded differences in which the variables running through the new dimensions are unconstrained. For instance, when starting from the BDS $\mathcal{B} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the BDS

$$\left\{ (x, y, z)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{B} \right\}.$$

Definition at line 2611 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.10 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project (dimension_type *m*) [inline]

Adds `m` new dimensions to the BDS and does not embed it in the new vector space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new BDS, which is defined by a system of bounded differences in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the BDS $\mathcal{B} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the BDS

$$\left\{ (x, y, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{B} \right\}.$$

Definition at line 2640 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.11 template<typename T > dimension_type Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension () const [inline]

Returns 0, if `*this` is empty; otherwise, returns the affine dimension of `*this`.

Definition at line 292 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::compute_predecessors(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::is_discrete().

11.4.4.12 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the affine image of `*this` under the function mapping variable `var` into the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned.

expr The numerator of the affine expression.

denominator The denominator of the affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this`.

Definition at line 3826 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.13 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` into the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted.

expr The numerator of the affine expression.

denominator The denominator of the affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this`.

Definition at line 4157 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.14 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 5811 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Bit_Matrix::ascii_dump(), Parma_Polyhedra_Library::DB_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::Status::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, and Parma_Polyhedra_Library::BD_Shape< T >::status.

11.4.4.15 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.4.4.16 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 5823 of file BD_Shape.templates.hh.

11.4.4.17 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact (const BD_Shape< T > & *y*) [inline, private]

If the upper bound of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Current implementation is based on a variant of Algorithm 4.1 in A. Bemporad, K. Fukuda, and F. D. Torrisi *Convexity Recognition of the Union of Polyhedra* Technical Report AUT00-13, ETH Zurich, 2000 tailored to the special case of BD shapes.

Note

It is assumed that `*this` and `y` are dimension-compatible; if the assumption does not hold, the behavior is undefined.

Definition at line 2060 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Constraint_System::empty(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::BD_Shape<

T >::is_empty(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_-Shape< T >::marked_empty(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_-Library::numer_denom(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_-Library::MIP_Problem::optimal_value(), Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::redundancy_-dbm, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_-Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::BD_Shape< T >::space_-dimension(), Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, Parma_Polyhedra_-Library::UNFEASIBLE_MIP_PROBLEM, and Parma_Polyhedra_Library::BD_Shape< T >::upper_-bound_assign().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign_if_exact().

11.4.4.18 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign (const BD_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the BHMZ05-widening of `*this` and `y`.

Parameters

y A BDS that *must* be contained in `*this`.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3071 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension(), Parma_Polyhedra_-Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_-Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_-Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_-Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::BD_Shape< T >::reset_-shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_-Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), and Parma_-Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign().

11.4.4.19 template<typename T > template<bool integer_upper_bound> bool Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact (const BD_Shape< T > & *y*) [inline, private]

If the upper bound of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Implementation for the rational (resp., integer) case is based on Theorem 5.2 (resp. Theorem 5.3) of [BHZ09b]. The Boolean template parameter `integer_upper_bound` allows for choosing between the rational and integer upper bound algorithms.

Note

It is assumed that `*this` and `y` are dimension-compatible; if the assumption does not hold, the behavior is undefined.
The integer case is only enabled if T is an integer datatype.

Definition at line 2214 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_COMPILE_TIME_CHECK, Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.20 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 4260 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.4.4.21 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 4502 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.4.4.22 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::bounds (const Linear_Expression & *expr*, bool *from_above*) const `[inline, private]`

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

from_above `true` if and only if the boundedness of interest is "from above".

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 1126 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(),

Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_-Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, Parma_-Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::MIP_-Problem::solve(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_-Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_-dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::bounds_from_above(), and Parma_-Polyhedra_Library::BD_Shape< T >::bounds_from_below().

11.4.4.23 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from above in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 362 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::bounds().

11.4.4.24 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from below in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 368 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::bounds().

11.4.4.25 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::can_recycle_congruence_systems () [inline, static]

Returns `false` indicating that this domain cannot recycle congruences.

Definition at line 263 of file BD_Shape.inlines.hh.

11.4.4.26 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::can_recycle_constraint_systems () [inline, static]

Returns `false` indicating that this domain cannot recycle constraints.

Definition at line 256 of file BD_Shape.inlines.hh.

11.4.4.27 template<typename T > template<typename Iterator > void Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign (const BD_Shape< T > & *y*, Iterator *first*, Iterator *last*, unsigned * *tp* = 0) [inline]

Assigns to *this the result of computing the CC76-extrapolation between *this and y.

Parameters

y A BDS that *must* be contained in *this.

first An iterator referencing the first stop-point.

last An iterator referencing one past the last stop-point.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if *this and y are dimension-incompatible.

Definition at line 2885 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.28 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign (const BD_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Assigns to *this the result of computing the CC76-extrapolation between *this and y.

Parameters

y A BDS that *must* be contained in *this.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if *this and y are dimension-incompatible.

Definition at line 781 of file BD_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign().

11.4.4.29 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` the result of restoring in `y` the constraints of `*this` that were lost by CC76-extrapolation applications.

Parameters

y A BDS that *must* contain `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Note

As was the case for widening operators, the argument `y` is meant to denote the value computed in the previous iteration step, whereas `*this` denotes the value computed in the current iteration step (in the *decreasing* iteration sequence). Hence, the call `x.CC76_narrowing_assign(y)` will assign to `x` the result of the computation $y \Delta x$.

Definition at line 3191 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.30 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders (std::vector< dimension_type > & *leaders*) const [inline, private]

Compute the leaders of zero-equivalence classes.

It is assumed that the BDS is not empty and shortest-path closed.

Definition at line 935 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::compute_predecessors(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.31 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::compute_predecessors (std::vector< dimension_type > & *predecessor*) const [inline, private]

Compute the (zero-equivalence classes) predecessor relation.

It is assumed that the BDS is not empty and shortest-path closed.

Definition at line 906 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), and Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign().

11.4.4.32 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

Exceptions

std::length_error Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 534 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.33 template<typename T > Congruence_System Parma_Polyhedra_Library::BD_Shape< T >::congruences () const [inline]

Returns a system of (equality) congruences satisfied by `*this`.

Definition at line 154 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences().

Referenced by Parma_Polyhedra_Library::Grid::Grid().

11.4.4.34 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::constrains (Variable *var*) const [inline]

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 878 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.35 template<typename T > Constraint_System Parma_Polyhedra_Library::BD_Shape< T >::constraints () const [inline]

Returns a system of constraints defining `*this`.

Definition at line 5393 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign().

11.4.4.36 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::contains (const BD_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 578 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(),

Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::strictly_contains().

11.4.4.37 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point () const [inline]

Returns `true` if and only if `*this` contains at least one integer point.

Definition at line 724 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.38 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds (dimension_type *v*, dimension_type *last_v*, const Linear_Expression & *sc_expr*, Coefficient_traits::const_reference *sc_den*, const N & *minus_lb_v*) [inline, private]

An helper function for the computation of affine relations.

For each dbm index `u` (less than or equal to `last_v` and different from `v`), deduce constraints of the form `u - v <= c`, starting from `minus_lb_v` which is a lower bound for `v`.

The shortest-path closure is able to deduce the constraint `u - v <= ub_u - lb_v`. We can be more precise if variable `u` played an active role in the computation of the lower bound for `v`, i.e., if the corresponding coefficient `q == sc_expr[u]/sc_den` is greater than zero. In particular:

- if `q >= 1`, then `u - v <= lb_u - lb_v`;
- if `0 < q < 1`, then `u - v <= (q*lb_u + (1-q)*ub_u) - lb_v`.

Definition at line 3307 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), and Parma_Polyhedra_Library::is_plus_infinity().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::refine().

11.4.4.39 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds (dimension_type *v*, dimension_type *last_v*, const Linear_Expression & *sc_expr*, Coefficient_traits::const_reference *sc_den*, const N & *ub_v*) [inline, private]

An helper function for the computation of affine relations.

For each dbm index u (less than or equal to last_v and different from v), deduce constraints of the form v - u <= c, starting from ub_v which is an upper bound for v.

The shortest-path closure is able to deduce the constraint v - u <= ub_v - lb_u. We can be more precise if variable u played an active role in the computation of the upper bound for v, i.e., if the corresponding coefficient q == sc_expr[u]/sc_den is greater than zero. In particular:

- if q >= 1, then v - u <= ub_v - ub_u;
- if 0 < q < 1, then v - u <= ub_v - (q*ub_u + (1-q)*lb_u).

Definition at line 3246 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), and Parma_Polyhedra_Library::is_plus_infinity().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::refine().

11.4.4.40 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::difference_assign (const BD_Shape< T > & *y*) [inline]

Assigns to *this the smallest BD shape containing the set difference of *this and y.

Exceptions

std::invalid_argument Thrown if *this and y are dimension-incompatible.

Definition at line 2317 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.41 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens *this by dropping some points with non-integer coordinates for the space dimensions corresponding to vars.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 5659 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.42 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 5639 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.43 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper (N & *elem*) [inline, private]

Definition at line 842 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), and Parma_Polyhedra_Library::Checked::Result.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points().

11.4.4.44 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension (Variable *var*, dimension_type *m*) [inline]

Creates m copies of the space dimension corresponding to var.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if var does not correspond to a dimension of the vector space.

std::length_error Thrown if adding m new space dimensions would cause the vector space to exceed dimension max_space_dimension().

If *this has space dimension n, with $n > 0$, and var has space dimension $k \leq n$, then the k-th space dimension is expanded to m new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 5551 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::max_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

11.4.4.45 template<typename T > memory_size_type Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by *this.

Definition at line 5835 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Bit_Matrix::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Matrix< T >::external_memory_in_bytes(), and Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::total_memory_in_bytes().

11.4.4.46 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*) [inline]

Folds the space dimensions in vars into dest.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 5594 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

11.4.4.47 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints (dimension_type *v*) [inline, private]

Removes all the constraints on row/column `v`.

Definition at line 3368 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::PLUS_INFINITY.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), and Parma_Polyhedra_Library::BD_Shape< T >::unconstrain().

11.4.4.48 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints (dimension_type *v*) [inline, private]

Removes all binary constraints on row/column `v`.

Definition at line 3379 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::PLUS_INFINITY.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image().

11.4.4.49 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::frequency (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const [inline]

Returns `true` if and only if there exist a unique value `val` such that `*this` saturates the equality `expr = val`.

Parameters

expr The linear expression for which the frequency is needed;

freq_n If `true` is returned, the value is set to 0; Present for interface compatibility with class Grid, where the frequency can have a non-zero value;

freq_d If `true` is returned, the value is set to 1;

val_n The numerator of `val`;

val_d The denominator of `val`;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `false` is returned, then `freq_n`, `freq_d`, `val_n` and `val_d` are left untouched.

Definition at line 769 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::normalize2(), Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.50 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the image of `*this` with respect to the affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression.

relsym The relation symbol.

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `relsym` is a strict relation symbol.

Definition at line 4967 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

11.4.4.51 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()) [inline]`

Assigns to `*this` the image of `*this` with respect to the affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine transfer function.

relsym The relation symbol.

expr The numerator of the right hand side affine expression.

denominator The denominator of the right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this` or if `relsym` is a strict relation symbol.

Definition at line 4570 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, PPL_DIRTY_TEMP_COEFFICIENT,

Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.52 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the preimage of `*this` with respect to the affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression.

relsym The relation symbol.

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `relsym` is a strict relation symbol.

Definition at line 5230 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

11.4.4.53 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the preimage of `*this` with respect to the affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine transfer function.

relsym The relation symbol.

expr The numerator of the right hand side affine expression.

denominator The denominator of the right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this` or if `relsym` is a strict relation symbol.

Definition at line 5152 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.54 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape (const Constraint_System & *cs*, BD_Shape< T > & *limiting_shape*) const [inline, private]

Adds to `limiting_shape` the bounded differences in `cs` that are satisfied by `*this`.

Definition at line 2958 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign().

11.4.4.55 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign (const BD_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the H79-widening between `*this` and `y`.

Parameters

y A BDS that *must* be contained in `*this`.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 798 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Polyhedron::H79_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::widening_assign().

11.4.4.56 template<typename T > int32_t Parma_Polyhedra_Library::BD_Shape< T >::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 836 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.57 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign (Variable *var*) const [inline, private]

Incrementally computes shortest-path closure, assuming that only constraints affecting variable `var` need to be considered.

Note

It is assumed that `*this`, which was shortest-path closed, has only been modified by adding constraints affecting variable `var`. If this assumption is not satisfied, i.e., if a non-redundant constraint not affecting variable `var` has been added, the behavior is undefined.

Definition at line 1813 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(),

Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::PLUS_INFINITY, and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.58 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::integer_upper_bound_assign_if_exact (const BD_Shape< T > & *y*) [inline]

If the *integer* upper bound of `*this` and `y` is exact, it is assigned to `*this` and `true` is returned; otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Note

The integer upper bound of two rational BDS is the smallest rational BDS containing all the integral points of the two arguments. This method requires that the coefficient type parameter `T` is an integral type.

Definition at line 721 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::is_integer(), PPL_COMPILE_TIME_CHECK, Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.59 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` the intersection of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2839 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.60 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_bounded () const [inline]

Returns `true` if and only if `*this` is a bounded BDS.

Definition at line 702 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.61 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_discrete () const [inline]

Returns `true` if and only if `*this` is discrete.

Definition at line 412 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension().

11.4.4.62 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from (const BD_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 637 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.63 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_empty () const [inline]

Returns `true` if and only if `*this` is an empty BDS.

Definition at line 355 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape<

T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.64 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced () const [inline, private]

Returns `true` if and only if `this->dbm` is shortest-path closed and `this->redundancy_dbm` correctly flags the redundant entries in `this->dbm`.

Definition at line 955 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign().

11.4.4.65 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_topologically_closed () const [inline]

Returns `true` if and only if `*this` is a topologically closed subset of the vector space.

Definition at line 406 of file BD_Shape.inlines.hh.

11.4.4.66 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::is_universe () const [inline]

Returns `true` if and only if `*this` is a universe BDS.

Definition at line 679 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.67 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign (const BD_Shape< T > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the BHMZ05-widening computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y A BDS that *must* be contained in `*this`.

cs The system of constraints used to improve the widened BDS.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible or if `cs` contains a strict inequality.

Definition at line 3141 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::BD_Shape< T >::throw_generic(), and Parma_Polyhedra_Library::UNIVERSE.

11.4.4.68 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign (const BD_Shape< T > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y A BDS that *must* be contained in `*this`.

cs The system of constraints used to improve the widened BDS.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible or if `cs` contains a strict inequality.

Definition at line 3020 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::BD_Shape< T >::throw_generic(), and Parma_Polyhedra_Library::UNIVERSE.

11.4.4.69 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::limited_H79_extrapolation_assign (const BD_Shape< T > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the H79-widening computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y A BDS that *must* be contained in `*this`.

cs The system of constraints used to improve the widened BDS.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible.

Definition at line 816 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

11.4.4.70 template<typename T > template<typename Partial_Function > void Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

Parameters

pfunc The partial function specifying the destiny of each dimension.

The template type parameter Partial_Function must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty co-domain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the co-domain of the partial function.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned.

The result is undefined if `pfunc` does not encode a partial function with the properties described in the specification of the mapping operator.

Definition at line 2771 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

11.4.4.71 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::marked_empty () const [inline, private]

Returns `true` if the BDS is known to be empty.

The return value `false` does not necessarily implies that `*this` is non-empty.

Definition at line 61 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::status, and Parma_Polyhedra_Library::BD_Shape< T >::Status::test_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::is_bounded(), Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::is_universe(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(),

Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.72 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed () const [inline, private]

Returns `true` if the system of bounded differences is known to be shortest-path closed.

The return value `false` does not necessarily implies that `this->dbm` is not shortest-path closed.

Definition at line 67 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::status, and Parma_Polyhedra_Library::BD_Shape< T >::Status::test_shortest_path_closed().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.73 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced () const [inline, private]

Returns `true` if the system of bounded differences is known to be shortest-path reduced.

The return value `false` does not necessarily implies that `this->dbm` is not shortest-path reduced.

Definition at line 73 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::status, and Parma_Polyhedra_Library::BD_Shape< T >::Status::test_shortest_path_reduced().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::operator=(), Parma_Polyhedra_Library::BD_Shape<

T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.74 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::marked_zero_dim_univ () const [inline, private]

Returns `true` if the BDS is the zero-dimensional universe.

Definition at line 55 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::status, and Parma_Polyhedra_Library::BD_Shape< T >::Status::test_zero_dim_univ().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape().

11.4.4.75 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*) const [inline, private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in * this;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `point` are left untouched.

Definition at line 1171 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::numer_denom(), Parma_Polyhedra_Library::MIP_Problem::optimal_value(), Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.76 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*, Generator & *g*) const [inline, private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in * this;

g When maximization or minimization succeeds, will be assigned a point or closure point where `expr` reaches the corresponding extremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `g` are left untouched.

Definition at line 1270 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, Parma_Polyhedra_Library::MIP_Problem::optimizing_point(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::maximize(), and Parma_Polyhedra_Library::BD_Shape< T >::minimize().

11.4.4.77 template<typename T > dimension_type Parma_Polyhedra_Library::BD_Shape< T >::max_space_dimension () [inline, static]

Returns the maximum space dimension that a BDS can handle.

Definition at line 46 of file BD_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension().

11.4.4.78 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value;

g When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d`, `maximum` and `g` are left untouched.

Definition at line 382 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::max_min().

11.4.4.79 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 374 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::max_min().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.4.4.80 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value;

g When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

Definition at line 398 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::max_min().

11.4.4.81 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 390 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::max_min().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.4.4.82 template<typename T > Congruence_System Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences () const [inline]

Returns a minimal system of (equality) congruences satisfied by `*this` with the same affine dimension as `*this`.

Definition at line 323 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Congruence_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::congruences().

11.4.4.83 template<typename T > Constraint_System Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints () const [inline]

Returns a minimized system of constraints defining `*this`.

Definition at line 5468 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::compute_leader_indices(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::constraints().

11.4.4.84 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::OK () const [inline]

Returns `true` if and only if `*this` satisfies all its invariants.

Definition at line 5842 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::BD_Shape< T >::Status::OK(), Parma_Polyhedra_Library::DB_Matrix< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.85 template<typename T > BD_Shape< T > & Parma_Polyhedra_Library::BD_Shape< T >::operator= (const BD_Shape< T > & *y*) [inline]

The assignment operator (`*this` and `y` can be dimension-incompatible).

Definition at line 326 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, and Parma_Polyhedra_Library::BD_Shape< T >::status.

11.4.4.86 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.4.4.87 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline, private]

Adds to the BDS the constraint $\mathrm{var} \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$.

Note that the coefficient of `var` in `expr` is null.

Definition at line 3441 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage().

11.4.4.88 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check (const Congruence & *cg*) [inline, private]

Uses the congruence `cg` to refine `*this`.

Parameters

cg The congruence to be added. Nontrivial proper congruences are ignored. Non BD equalities are ignored.

Warning

If `cg` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 238 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Congruence::space_dimension().

11.4.4.89 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check (const Constraint & *c*) [inline, private]

Uses the constraint `c` to refine `*this`.

Parameters

c The constraint to be added. Non BD constraints are ignored.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 474 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint(), and Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints().

11.4.4.90 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence (const Congruence & *cg*) [inline]

Uses a copy of congruence `cg` to refine the system of bounded differences of `*this`.

Parameters

cg The congruence. If it is not a bounded difference equality, it will be ignored.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 213 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.91 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Uses a copy of the congruences in `cgs` to refine the system of bounded differences defining `*this`.

Parameters

cgs The congruence system to be used. Congruences that are not bounded difference equalities are ignored.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 225 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape().

11.4.4.92 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint (const Constraint & *c*) [inline]

Uses a copy of constraint `c` to refine the system of bounded differences defining `*this`.

Parameters

c The constraint. If it is not a bounded difference, it will be ignored.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 188 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.93 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints (const Constraint_System & *cs*) [inline]

Uses a copy of the constraints in `cs` to refine the system of bounded differences defining `*this`.

Parameters

cs The constraint system to be used. Constraints that are not bounded differences are ignored.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 200 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape().

11.4.4.94 template<typename T > Poly_Gen_Relation Parma_Polyhedra_Library::BD_Shape< T >::relation_with (const Generator & *g*) const [inline]

Returns the relations holding between `*this` and the generator `g`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible.

Definition at line 1651 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Coefficient_zero(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.95 template<typename T > Poly_Con_Relation Parma_Polyhedra_Library::BD_Shape< T >::relation_with (const Congruence & *cg*) const [inline]

Returns the relations holding between `*this` and the congruence `cg`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 1316 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::maximize(), Parma_Polyhedra_Library::BD_Shape< T >::minimize(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.96 template<typename T > Poly_Con_Relation Parma_Polyhedra_Library::BD_Shape< T >::relation_with (const Constraint & *c*) const [inline]

Returns the relations holding between `*this` and the constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 1407 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Constraint::is_nonstrict_inequality(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::maximize(), Parma_Polyhedra_Library::BD_Shape< T >::minimize(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.4.4.97 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions (dimension_type *new_dimension*) [inline]

Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimension` is greater than the space dimension of `*this`.

Definition at line 733 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), and Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions().

11.4.4.98 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions (const Variables_Set & *vars*) [inline]

Removes all the specified dimensions.

Parameters

vars The set of Variable objects corresponding to the dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 2686 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions().

11.4.4.99 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed () [inline, private]

Marks `*this` as possibly not shortest-path closed.

Definition at line 103 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.100 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced () [inline, private]

Marks `*this` as possibly not shortest-path reduced.

Definition at line 109 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_shortest_path_reduced(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.101 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::set_empty () [inline, private]

Turns `*this` into an empty BDS.

Definition at line 85 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::set_empty(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), and Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign().

11.4.4.102 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed () [inline, private]

Marks `*this` as shortest-path closed.

Definition at line 91 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::set_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), and Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign().

11.4.4.103 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_reduced () [inline, private]

Marks `*this` as shortest-path closed.

Definition at line 97 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::set_shortest_path_reduced(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign().

11.4.4.104 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ () [inline, private]

Turns *this into an zero-dimensional universe BDS.

Definition at line 79 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::Status::set_zero_dim_univ(), and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.105 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign () const [inline, private]

Assigns to this->dbm its shortest-path closure.

Definition at line 1755 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::is_bounded(), Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_

Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.4.106 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign () const [inline, private]

Assigns to `this->dbm` its shortest-path closure and records into `this->redundancy_dbm` which of the entries in `this->dbm` are redundant.

Definition at line 1923 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Bit_Matrix::clear(), Parma_Polyhedra_Library::Bit_Row::clear(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leader_indices(), Parma_Polyhedra_Library::BD_Shape< T >::compute_predecessors(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), and Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign().

11.4.4.107 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 2387 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::redundancy_-

dbm, Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.4.4.108 template<typename T > dimension_type Parma_Polyhedra_Library::BD_Shape< T >::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 349 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, and Parma_Polyhedra_Library::DB_Matrix< T >::num_rows().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::hash_code(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::is_bounded(), Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::is_universe(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::refine_-

with_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign_if_exact().

11.4.4.109 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::strictly_contains (const BD_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` strictly contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 701 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::contains().

11.4.4.110 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::swap (BD_Shape< T > & *y*) [inline]

Swaps `*this` with `y` (`*this` and `y` can be dimension-incompatible).

Definition at line 341 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm, and Parma_Polyhedra_Library::BD_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), and Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign().

11.4.4.111 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, const char * *name_row*, const Linear_Expression & *y*) const [inline, private]

Definition at line 6000 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_-Library::BD_Shape< T >::space_dimension().

11.4.4.112 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, const Generator & *g*) const [inline, private]

Definition at line 5977 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Generator::space_dimension(), and Parma_Polyhedra_-Library::BD_Shape< T >::space_dimension().

11.4.4.113 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, const Congruence & *cg*) const [inline, private]

Definition at line 5966 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_-Library::BD_Shape< T >::space_dimension().

11.4.4.114 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, const Constraint & *c*) const [inline, private]

Definition at line 5955 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_-Library::BD_Shape< T >::space_dimension().

11.4.4.115 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, dimension_type *required_dim*) const [inline, private]

Definition at line 5944 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.4.116 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible (const char * *method*, const BD_Shape< T > & *x*) const [inline, private]

Definition at line 5933 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign_if_exact().

11.4.4.117 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_expression_too_complex (const char * *method*, const Linear_Expression & *e*) [inline, static, private]

Definition at line 5988 of file BD_Shape.templates.hh.

11.4.4.118 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::throw_generic (const char * *method*, const char * *reason*) [inline, static, private]

Definition at line 6013 of file BD_Shape.templates.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences(), and Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints().

11.4.4.119 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 686 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.4.4.120 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 418 of file BD_Shape.inlines.hh.

11.4.4.121 template<typename T > memory_size_type Parma_Polyhedra_Library::BD_Shape< T >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 830 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes().

11.4.4.122 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::unconstrain (const Variables_Set & *vars*) [inline]

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 3412 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.123 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::unconstrain (Variable *var*) [inline]

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 3390 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.124 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign (const BD_Shape< T > & *y*) [inline]

Assigns to `*this` the smallest BDS containing the union of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2021 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::BD_Shape< T >::difference_assign().

11.4.4.125 template<typename T > bool Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign_if_exact (const BD_Shape< T > & *y*) [inline]

If the upper bound of `*this` and `y` is exact, it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 708 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.4.4.126 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::widening_assign (const BD_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Same as H79_widening_assign(y, tp).

Definition at line 810 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign().

11.4.4.127 template<typename T > void Parma_Polyhedra_Library::BD_Shape< T >::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true) [inline]

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system whose variables are contained in `vars`. If `*pcs` depends on variables not in `vars`, the behavior is undefined. When non-null, the pointed-to constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in `*pcs`.

complexity_threshold A precision parameter of the wrapping operator: higher values result in possibly improved precision.

wrap_individually `true` if the dimensions should be wrapped individually (something that results in much greater efficiency to the detriment of precision).

Exceptions

std::invalid_argument Thrown if `*pcs` is dimension-incompatible with `vars`, or if `*this` is dimension-incompatible `vars` or with `*pcs`.

Definition at line 766 of file BD_Shape.inlines.hh.

11.4.5 Friends And Related Function Documentation

11.4.5.1 template<typename T> void compute_leader_indices (const std::vector< dimension_type > & *predecessor*, std::vector< dimension_type > & *indices*) [related]

Extracts leader indices from the predecessor relation.

Definition at line 87 of file BD_Shape.cc.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign().

11.4.5.2 template<typename Temp , typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 528 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.5.3 template<typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 569 of file BD_Shape.inlines.hh.

11.4.5.4 template<typename T> bool extract_bounded_difference (const Constraint & *c*, dimension_type *c_space_dim*, dimension_type & *c_num_vars*, dimension_type & *c_first_var*, dimension_type & *c_second_var*, Coefficient & *c_coeff*) [related]

Decodes the constraint `c` as a bounded difference.

Returns

`true` if the constraint `c` is a bounded difference; `false` otherwise.

Parameters

c The constraint to be decoded.

c_space_dim The space dimension of the constraint `c` (it is *assumed* to match the actual space dimension of `c`).

c_num_vars If `true` is returned, then it will be set to the number of variables having a non-zero coefficient. The only legal values will therefore be 0, 1 and 2.

c_first_var If `true` is returned and if `c_num_vars` is not set to 0, then it will be set to the index of the first variable having a non-zero coefficient in `c`.

c_second_var If `true` is returned and if `c_num_vars` is set to 2, then it will be set to the index of the second variable having a non-zero coefficient in `c`.

c_coeff If `true` is returned and if `c_num_vars` is not set to 0, then it will be set to the value of the first non-zero coefficient in `c`.

Definition at line 33 of file BD_Shape.cc.

References Parma_Polyhedra_Library::Constraint::coefficient(), and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.4.5.5 template<typename Temp , typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between x and y.

If the L_∞ distance between x and y is defined, stores an approximation of it into r and returns true; returns false otherwise.

The direction of the approximation is specified by dir.

All computations are performed using the temporary variables tmp0, tmp1 and tmp2.

Definition at line 593 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.5.6 template<typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the L_∞ distance between x and y.

If the L_∞ distance between x and y is defined, stores an approximation of it into r and returns true; returns false otherwise.

The direction of the approximation is specified by dir.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the L_∞ distance between x and y is defined, stores an approximation of it into r and returns true; returns false otherwise.

The direction of the approximation is specified by dir.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 634 of file BD_Shape.inlines.hh.

11.4.5.7 template<typename T > bool operator!= (const BD_Shape< T > & *x*, const BD_Shape< T > & *y*) [related]

Returns true if and only if x and y aren't the same BDS.

Note that x and y may be dimension-incompatible shapes: in this case, the value true is returned.

Definition at line 456 of file BD_Shape.inlines.hh.

11.4.5.8 template<typename T > std::ostream & operator<< (std::ostream & *s*, const BD_Shape< T > & *c*) [related]

Output operator.

Writes a textual representation of bds on s: false is written if bds is an empty polyhedron; true is

written if `bds` is the universe polyhedron; a system of constraints defining `bds` is written otherwise, all constraints separated by ", ".

Definition at line 5702 of file BD_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse(), and Parma_Polyhedra_Library::is_plus_infinity().

11.4.5.9 template<typename T> bool operator== (const BD_Shape< T > & *x*, const BD_Shape< T > & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same BDS.

Note that `x` and `y` may be dimension-incompatible shapes: in this case, the value `false` is returned.

Definition at line 424 of file BD_Shape.inlines.hh.

11.4.5.10 template<typename T> friend class Parma_Polyhedra_Library::BD_Shape [friend]

Definition at line 1800 of file BD_Shape.defs.hh.

11.4.5.11 template<typename T> friend class Parma_Polyhedra_Library::Box [friend]

Definition at line 1801 of file BD_Shape.defs.hh.

11.4.5.12 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< U > & *x*, const BD_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.4.5.13 template<typename T> std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const BD_Shape< T > & *c*) [friend]

11.4.5.14 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< U > & *x*, const BD_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.4.5.15 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< U > & *x*, const BD_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.4.5.16 template<typename Temp , typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 463 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.4.5.17 template<typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const BD_Shape< T > & *x*, const BD_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 504 of file BD_Shape.inlines.hh.

11.4.5.18 template<typename T > void swap (Parma_Polyhedra_Library::BD_Shape< T > & *x*, Parma_Polyhedra_Library::BD_Shape< T > & *y*) [related]

Specializes `std::swap`.

Definition at line 858 of file BD_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::swap().

11.4.6 Member Data Documentation

11.4.6.1 template<typename T> DB_Matrix<N> Parma_Polyhedra_Library::BD_Shape< T >::dbm [private]

The matrix representing the system of bounded differences.

Definition at line 1804 of file BD_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_dbm_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::contains(), Parma_Polyhedra_Library::BD_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::BD_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::BD_Shape< T >::is_bounded(), Parma_Polyhedra_Library::BD_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::is_universe(), Parma_Polyhedra_Library::BD_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::operator=(), Parma_Polyhedra_Library::BD_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), Parma_Polyhedra_

Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::BD_Shape< T >::space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::swap(), and Parma_Polyhedra_Library::BD_Shape< T >::upper_bound_assign().

11.4.6.2 template<typename T> Bit_Matrix Parma_Polyhedra_Library::BD_Shape< T >::redundancy_dbm [private]

A matrix indicating which constraints are redundant.

Definition at line 1814 of file BD_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::BD_Shape< T >::BHZ09_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::BD_Shape< T >::is_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::operator=(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::BD_Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

11.4.6.3 template<typename T> Status Parma_Polyhedra_Library::BD_Shape< T >::status [private]

The status flags to keep track of the internal state.

Definition at line 1811 of file BD_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::marked_zero_dim_univ(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::operator=(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed(), Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_reduced(), Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ(), and Parma_Polyhedra_Library::BD_Shape< T >::swap().

The documentation for this class was generated from the following files:

- BD_Shape.defs.hh
- BD_Shape.inlines.hh
- BD_Shape.templates.hh
- BD_Shape.cc

11.5 Parma_Polyhedra_Library::BHRZ03_Certificate Class Reference

The convergence certificate for the BHRZ03 widening operator.

`#include <BHRZ03_Certificate.defs.hh>`

Classes

- struct Compare

  A total ordering on BHRZ03 certificates.

Public Member Functions

- BHRZ03_Certificate ()

  Default constructor.

- BHRZ03_Certificate (const Polyhedron &ph)

  Constructor: computes the certificate for `ph`*.*

- BHRZ03_Certificate (const BHRZ03_Certificate &y)

  Copy constructor.

- ~BHRZ03_Certificate ()

  Destructor.

- int compare (const BHRZ03_Certificate &y) const

  The comparison function for certificates.

- int compare (const Polyhedron &ph) const

  Compares `*this` *with the certificate for polyhedron* `ph`*.*

- bool is_stabilizing (const Polyhedron &ph) const

  Returns `true` *if and only if the certificate for polyhedron* `ph` *is strictly smaller than* `*this`*.*

- bool OK () const

  Check if gathered information is meaningful.

Private Attributes

- dimension_type affine_dim

  Affine dimension of the polyhedron.

- dimension_type lin_space_dim

  Dimension of the lineality space of the polyhedron.

- dimension_type num_constraints

  Cardinality of a non-redundant constraint system for the polyhedron.

- dimension_type num_points

  Number of non-redundant points in a generator system for the polyhedron.

- std::vector< dimension_type > num_rays_null_coord

  A vector containing, for each index '0 <= i < space_dim', the number of non-redundant rays in a generator system of the polyhedron having exactly 'i' null coordinates.

11.5.1 Detailed Description

The convergence certificate for the BHRZ03 widening operator. Convergence certificates are used to instantiate the BHZ03 framework so as to define widening operators for the finite powerset domain.

Note

Each convergence certificate has to be used together with a compatible widening operator. In particular, BHRZ03_Certificate can certify the convergence of both the BHRZ03 and the H79 widenings.

Definition at line 43 of file BHRZ03_Certificate.defs.hh.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate () [inline]

Default constructor.

Definition at line 30 of file BHRZ03_Certificate.inlines.hh.

References OK().

11.5.2.2 Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate (const Polyhedron & *ph*)

Constructor: computes the certificate for `ph`.

Definition at line 35 of file BHRZ03_Certificate.cc.

References affine_dim, Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed(), lin_space_dim, Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::minimized_constraints(), Parma_Polyhedra_Library::Polyhedron::minimized_generators(), num_constraints, num_points, num_rays_null_coord, OK(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, and Parma_Polyhedra_Library::Polyhedron::space_dimension().

11.5.2.3 Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate (const BHRZ03_Certificate & *y*) [inline]

Copy constructor.

Definition at line 38 of file BHRZ03_Certificate.inlines.hh.

11.5.2.4 Parma_Polyhedra_Library::BHRZ03_Certificate::~BHRZ03_Certificate () [inline]

Destructor.

Definition at line 45 of file BHRZ03_Certificate.inlines.hh.

11.5.3 Member Function Documentation

11.5.3.1 int Parma_Polyhedra_Library::BHRZ03_Certificate::compare (const Polyhedron & *ph*) const

Compares `*this` with the certificate for polyhedron `ph`.

Definition at line 130 of file BHRZ03_Certificate.cc.

References affine_dim, Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed(), lin_space_dim, Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::minimized_constraints(), Parma_Polyhedra_Library::Polyhedron::minimized_generators(), num_constraints, num_points, num_rays_null_coord, Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::Polyhedron::space_dim, and Parma_Polyhedra_Library::Polyhedron::space_dimension().

11.5.3.2 int Parma_Polyhedra_Library::BHRZ03_Certificate::compare (const BHRZ03_Certificate & *y*) const

The comparison function for certificates.

Returns

-1, 0 or 1 depending on whether `*this` is smaller than, equal to, or greater than `y`, respectively.

Compares `*this` with `y`, using a total ordering which is a refinement of the limited growth ordering relation for the BHRZ03 widening.

Definition at line 107 of file BHRZ03_Certificate.cc.

References affine_dim, lin_space_dim, num_constraints, num_points, num_rays_null_coord, and OK().

Referenced by is_stabilizing(), and Parma_Polyhedra_Library::BHRZ03_Certificate::Compare::operator()().

11.5.3.3 bool Parma_Polyhedra_Library::BHRZ03_Certificate::is_stabilizing (const Polyhedron & *ph*) const [inline]

Returns `true` if and only if the certificate for polyhedron `ph` is strictly smaller than `*this`.

Definition at line 49 of file BHRZ03_Certificate.inlines.hh.

References compare().

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), and Parma_Polyhedra_Library::Polyhedron::BHRZ03_widening_assign().

11.5.3.4 bool Parma_Polyhedra_Library::BHRZ03_Certificate::OK () const

Check if gathered information is meaningful.

Definition at line 252 of file BHRZ03_Certificate.cc.

References affine_dim, lin_space_dim, num_constraints, num_points, and num_rays_null_coord.

Referenced by BHRZ03_Certificate(), and compare().

11.5.4 Member Data Documentation

11.5.4.1 dimension_type Parma_Polyhedra_Library::BHRZ03_Certificate::affine_dim [private]

Affine dimension of the polyhedron.

Definition at line 98 of file BHRZ03_Certificate.defs.hh.

Referenced by BHRZ03_Certificate(), compare(), and OK().

11.5.4.2 dimension_type Parma_Polyhedra_Library::BHRZ03_Certificate::lin_space_dim [private]

Dimension of the lineality space of the polyhedron.

Definition at line 100 of file BHRZ03_Certificate.defs.hh.

Referenced by BHRZ03_Certificate(), compare(), and OK().

11.5.4.3 dimension_type Parma_Polyhedra_Library::BHRZ03_Certificate::num_constraints [private]

Cardinality of a non-redundant constraint system for the polyhedron.

Definition at line 102 of file BHRZ03_Certificate.defs.hh.

Referenced by BHRZ03_Certificate(), compare(), and OK().

11.5.4.4 dimension_type Parma_Polyhedra_Library::BHRZ03_Certificate::num_points [private]

Number of non-redundant points in a generator system for the polyhedron.

Definition at line 107 of file BHRZ03_Certificate.defs.hh.

Referenced by BHRZ03_Certificate(), compare(), and OK().

11.5.4.5 std::vector<dimension_type> Parma_Polyhedra_Library::BHRZ03_Certificate::num_rays_null_coord [private]

A vector containing, for each index '0 <= i < space_dim', the number of non-redundant rays in a generator system of the polyhedron having exactly 'i' null coordinates.

Definition at line 113 of file BHRZ03_Certificate.defs.hh.

Referenced by BHRZ03_Certificate(), compare(), and OK().

The documentation for this class was generated from the following files:

- BHRZ03_Certificate.defs.hh
- BHRZ03_Certificate.inlines.hh
- BHRZ03_Certificate.cc

11.6 Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign > Class Template Reference

A function adapter for the Determinate class.

`#include <Determinate.defs.hh>`

Public Member Functions

- Binary_Operator_Assign_Lifter (Binary_Operator_Assign op_assign)

  Explicit unary constructor.

- void operator() (Determinate &x, const Determinate &y) const

  Function-application operator.

Private Attributes

- Binary_Operator_Assign op_assign_

  The function object to be lifted.

11.6.1 Detailed Description

template<typename PSET>template<typename Binary_Operator_Assign> class Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >

A function adapter for the Determinate class. It lifts a Binary_Operator_Assign function object, taking arguments of type PSET, producing the corresponding function object taking arguments of type Determinate<PSET>.

The template parameter Binary_Operator_Assign is supposed to implement an *apply and assign* function, i.e., a function having signature `void foo(PSET& x, const PSET& y)` that applies an operator to `x` and `y` and assigns the result to `x`. For instance, such a function object is obtained by `std::mem_fun_ref(&C_Polyhedron::intersection_assign)`.

Definition at line 214 of file Determinate.defs.hh.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 template<typename PSET > template<typename Binary_Operator_Assign > Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::Binary_Operator_Assign_Lifter (Binary_Operator_Assign *op_assign*) [inline, explicit]

Explicit unary constructor.

Definition at line 275 of file Determinate.inlines.hh.

11.6.3 Member Function Documentation

11.6.3.1 template<typename PSET > template<typename Binary_Operator_Assign > void Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::operator() (Determinate & *x*, const Determinate & *y*) const [inline]

Function-application operator.

Definition at line 283 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::op_assign_, and Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.6.4 Member Data Documentation

11.6.4.1 template<typename PSET> template<typename Binary_Operator_Assign> Binary_Operator_Assign Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::op_assign_ [private]

The function object to be lifted.

Definition at line 225 of file Determinate.defs.hh.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::operator()().

The documentation for this class was generated from the following files:

- Determinate.defs.hh
- Determinate.inlines.hh

11.7 Parma_Polyhedra_Library::Bit_Matrix Class Reference

A matrix of bits.

`#include <Bit_Matrix.defs.hh>`

Classes

- struct Bit_Row_Less_Than

  Ordering predicate (used when implementing the sort algorithm).

Public Member Functions

- Bit_Matrix ()

  Default constructor.

- Bit_Matrix (dimension_type n_rows, dimension_type n_columns)

  Construct a bit matrix with `n_rows` rows and `n_columns` columns.

- Bit_Matrix (const Bit_Matrix &y)

  Copy constructor.

- ~Bit_Matrix ()

  Destructor.

- Bit_Matrix & operator= (const Bit_Matrix &y)

  Assignment operator.

- void swap (Bit_Matrix &y)

  Swaps `*this` with `y`.

- Bit_Row & operator[ ] (dimension_type k)

  Subscript operator.

- const Bit_Row & operator[ ] (dimension_type k) const

  Constant subscript operator.

- void clear ()

Clears the matrix deallocating all its rows.

- void transpose ()

  Transposes the matrix.

- void transpose_assign (const Bit_Matrix &y)

  Makes `*this` *a transposed copy of* `y`.

- dimension_type num_columns () const

  Returns the number of columns of `*this`.

- dimension_type num_rows () const

  Returns the number of rows of `*this`.

- void sort_rows ()

  Sorts the rows and removes duplicates.

- bool sorted_contains (const Bit_Row &row) const

  Looks for `row` *in* `*this`*, which is assumed to be sorted.*

- void add_recycled_row (Bit_Row &row)

  Adds `row` *to* `*this`.

- void rows_erase_to_end (dimension_type first_to_erase)

  Erases the rows from the `first_to_erase` *-th to the last one.*

- void columns_erase_to_end (dimension_type first_to_erase)

  Erases the columns from the `first_to_erase` *-th to the last one.*

- void resize (dimension_type new_n_rows, dimension_type new_n_columns)

  Resizes the matrix copying the old contents.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool check_sorted () const

  Checks whether `*this` *is sorted. It does NOT check for duplicates.*

Static Public Member Functions

- static dimension_type max_num_rows ()

  Returns the maximum number of rows of a Bit_Matrix.

Private Attributes

- std::vector< Bit_Row > rows

  Contains the rows of the matrix.

- dimension_type row_size

  Size of the initialized part of each row.

Friends

- void Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat (Bit_Matrix &sat)

Related Functions

(Note that these are not member functions.)

- bool operator== (const Bit_Matrix &x, const Bit_Matrix &y)

  Returns `true` *if and only if* `x` *and* `y` *are equal.*

- bool operator!= (const Bit_Matrix &x, const Bit_Matrix &y)

  Returns `true` *if and only if* `x` *and* `y` *are not equal.*

- void swap (Parma_Polyhedra_Library::Bit_Matrix &x, Parma_Polyhedra_Library::Bit_Matrix &y)

  Specializes `std::swap`.

11.7.1 Detailed Description

A matrix of bits.

Definition at line 37 of file Bit_Matrix.defs.hh.

11.7.2 Constructor & Destructor Documentation

11.7.2.1 Parma_Polyhedra_Library::Bit_Matrix::Bit_Matrix () [inline]

Default constructor.

Definition at line 33 of file Bit_Matrix.inlines.hh.

11.7.2.2 Parma_Polyhedra_Library::Bit_Matrix::Bit_Matrix (dimension_type *n_rows*, dimension_type *n_columns*) [inline]

Construct a bit matrix with `n_rows` rows and `n_columns` columns.

Definition at line 44 of file Bit_Matrix.inlines.hh.

11.7.2.3 Parma_Polyhedra_Library::Bit_Matrix::Bit_Matrix (const Bit_Matrix & *y*) [inline]

Copy constructor.

Definition at line 51 of file Bit_Matrix.inlines.hh.

11.7.2.4 Parma_Polyhedra_Library::Bit_Matrix::~Bit_Matrix () [inline]

Destructor.

Definition at line 57 of file Bit_Matrix.inlines.hh.

11.7.3 Member Function Documentation

11.7.3.1 void Parma_Polyhedra_Library::Bit_Matrix::add_recycled_row (Bit_Row & *row*)

Adds `row` to `*this`.

Parameters

row The row whose implementation will be recycled.

The only thing that can be done with `row` upon return is destruction.

Definition at line 67 of file Bit_Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), max_num_rows(), OK(), rows, and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion().

11.7.3.2 void Parma_Polyhedra_Library::Bit_Matrix::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 152 of file Bit_Matrix.cc.

References num_columns(), and num_rows().

11.7.3.3 void Parma_Polyhedra_Library::Bit_Matrix::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Polyhedron::ascii_dump(), and Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump().

11.7.3.4 bool Parma_Polyhedra_Library::Bit_Matrix::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 167 of file Bit_Matrix.cc.

References clear().

11.7.3.5 bool Parma_Polyhedra_Library::Bit_Matrix::check_sorted () const

Checks whether `*this` is sorted. It does NOT check for duplicates.

Definition at line 232 of file Bit_Matrix.cc.

References num_rows().

Referenced by sorted_contains().

11.7.3.6 void Parma_Polyhedra_Library::Bit_Matrix::clear () [inline]

Clears the matrix deallocating all its rows.

Definition at line 108 of file Bit_Matrix.inlines.hh.

References row_size, rows, and swap().

Referenced by ascii_load(), Parma_Polyhedra_Library::Polyhedron::set_empty(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::Polyhedron::update_sat_c(), and Parma_Polyhedra_Library::Polyhedron::update_sat_g().

11.7.3.7 void Parma_Polyhedra_Library::Bit_Matrix::columns_erase_to_end (dimension_type *first_to_erase*) [inline]

Erases the columns from the `first_to_erase` -th to the last one.

Definition at line 71 of file Bit_Matrix.inlines.hh.

References OK(), and row_size.

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion().

11.7.3.8 PPL::memory_size_type Parma_Polyhedra_Library::Bit_Matrix::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Definition at line 197 of file Bit_Matrix.cc.

References num_rows(), and rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::external_memory_in_bytes(), Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes(), and total_memory_in_bytes().

11.7.3.9 dimension_type Parma_Polyhedra_Library::Bit_Matrix::max_num_rows () [inline, static]

Returns the maximum number of rows of a Bit_Matrix.

Definition at line 39 of file Bit_Matrix.inlines.hh.

Referenced by add_recycled_row(), and resize().

11.7.3.10 dimension_type Parma_Polyhedra_Library::Bit_Matrix::num_columns () const [inline]

Returns the number of columns of `*this`.

Definition at line 98 of file Bit_Matrix.inlines.hh.

References row_size.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), ascii_dump(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::OK(), operator==(), Parma_Polyhedra_Library::Polyhedron::simplify(), transpose(), and transpose_assign().

11.7.3.11 dimension_type Parma_Polyhedra_Library::Bit_Matrix::num_rows () const [inline]

Returns the number of rows of `*this`.

Definition at line 103 of file Bit_Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), ascii_dump(), check_sorted(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::conversion(), external_memory_in_bytes(), Parma_Polyhedra_Library::Polyhedron::OK(), OK(), operator==(), resize(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), transpose(), and transpose_assign().

11.7.3.12 bool Parma_Polyhedra_Library::Bit_Matrix::OK () const

Checks if all the invariants are satisfied.

Definition at line 205 of file Bit_Matrix.cc.

References Parma_Polyhedra_Library::Bit_Row::last(), num_rows(), Parma_Polyhedra_Library::Bit_Row::OK(), and row_size.

Referenced by add_recycled_row(), columns_erase_to_end(), Parma_Polyhedra_Library::Polyhedron::OK(), operator=(), resize(), rows_erase_to_end(), sort_rows(), transpose(), and transpose_assign().

11.7.3.13 PPL::Bit_Matrix & Parma_Polyhedra_Library::Bit_Matrix::operator= (const Bit_Matrix & *y*)

Assignment operator.

Definition at line 37 of file Bit_Matrix.cc.

References OK(), row_size, and rows.

11.7.3.14 const Bit_Row & Parma_Polyhedra_Library::Bit_Matrix::operator[ ] (dimension_type *k*) const [inline]

Constant subscript operator.

Definition at line 92 of file Bit_Matrix.inlines.hh.

References rows.

11.7.3.15 Bit_Row & Parma_Polyhedra_Library::Bit_Matrix::operator[ ] (dimension_type *k*) [inline]

Subscript operator.

Definition at line 86 of file Bit_Matrix.inlines.hh.

References rows.

11.7.3.16 void Parma_Polyhedra_Library::Bit_Matrix::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.7.3.17 void Parma_Polyhedra_Library::Bit_Matrix::resize (dimension_type *new_n_rows*, dimension_type *new_n_columns*)

Resizes the matrix copying the old contents.

Definition at line 116 of file Bit_Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), max_num_rows(), num_rows(), OK(), row_size, rows, and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::update_sat_c(), and Parma_Polyhedra_Library::Polyhedron::update_sat_g().

11.7.3.18 void Parma_Polyhedra_Library::Bit_Matrix::rows_erase_to_end (dimension_type *first_to_erase*) [inline]

Erases the rows from the `first_to_erase` -th to the last one.

Definition at line 61 of file Bit_Matrix.inlines.hh.

References OK(), and rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify(), and Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat().

11.7.3.19 void Parma_Polyhedra_Library::Bit_Matrix::sort_rows ()

Sorts the rows and removes duplicates.

Definition at line 45 of file Bit_Matrix.cc.

References OK(), and rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::select_H79_constraints().

11.7.3.20 bool Parma_Polyhedra_Library::Bit_Matrix::sorted_contains (const Bit_Row & *row*) const [inline]

Looks for `row` in `*this`, which is assumed to be sorted.

Returns

`true` if `row` belongs to `*this`, false otherwise.

Parameters

row The row that will be searched for in the matrix.

Given a sorted bit matrix (this ensures better efficiency), tells whether it contains the given row.

Definition at line 126 of file Bit_Matrix.inlines.hh.

References check_sorted(), and rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::select_H79_constraints().

11.7.3.21 void Parma_Polyhedra_Library::Bit_Matrix::swap (Bit_Matrix & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 80 of file Bit_Matrix.inlines.hh.

References row_size, and rows.

Referenced by add_recycled_row(), clear(), Parma_Polyhedra_Library::Polyhedron::conversion(), resize(), swap(), transpose(), and transpose_assign().

11.7.3.22 memory_size_type Parma_Polyhedra_Library::Bit_Matrix::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 115 of file Bit_Matrix.inlines.hh.

References external_memory_in_bytes().

11.7.3.23 void Parma_Polyhedra_Library::Bit_Matrix::transpose ()

Transposes the matrix.

Definition at line 91 of file Bit_Matrix.cc.

References num_columns(), num_rows(), OK(), and swap().

11.7.3.24 void Parma_Polyhedra_Library::Bit_Matrix::transpose_assign (const Bit_Matrix & *y*)

Makes `*this` a transposed copy of `y`.

Definition at line 104 of file Bit_Matrix.cc.

References num_columns(), num_rows(), OK(), and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_-Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints(), Parma_Polyhedra_-Library::Polyhedron::obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_-Library::Polyhedron::obtain_sorted_generators(), Parma_Polyhedra_Library::Polyhedron::obtain_-sorted_generators_with_sat_g(), Parma_Polyhedra_Library::Polyhedron::process_pending_-constraints(), Parma_Polyhedra_Library::Polyhedron::process_pending_generators(), Parma_-Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), and Parma_Polyhedra_-Library::Polyhedron::strongly_minimize_generators().

11.7.4 Friends And Related Function Documentation

11.7.4.1 bool operator!= (const Bit_Matrix & *x*, const Bit_Matrix & *y*) [related]

Returns `true` if and only if `x` and `y` are not equal.

Definition at line 134 of file Bit_Matrix.inlines.hh.

11.7.4.2 bool operator== (const Bit_Matrix & *x*, const Bit_Matrix & *y*) [related]

Returns `true` if and only if `x` and `y` are equal.

Definition at line 243 of file Bit_Matrix.cc.

References num_columns(), and num_rows().

11.7.4.3 void Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat (Bit_Matrix & *sat*) [friend]

11.7.4.4 void swap (Parma_Polyhedra_Library::Bit_Matrix & *x*, Parma_Polyhedra_-Library::Bit_Matrix & *y*) [related]

Specializes `std::swap`.

Definition at line 145 of file Bit_Matrix.inlines.hh.

References swap().

11.7.5 Member Data Documentation

11.7.5.1 dimension_type Parma_Polyhedra_Library::Bit_Matrix::row_size [private]

Size of the initialized part of each row.

Definition at line 143 of file Bit_Matrix.defs.hh.

Referenced by clear(), columns_erase_to_end(), num_columns(), OK(), operator=(), resize(), and swap().

11.7.5.2 std::vector<Bit_Row> Parma_Polyhedra_Library::Bit_Matrix::rows [private]

Contains the rows of the matrix.

Definition at line 140 of file Bit_Matrix.defs.hh.

Referenced by add_recycled_row(), clear(), external_memory_in_bytes(), num_rows(), operator=(), operator[ ](), resize(), rows_erase_to_end(), sort_rows(), sorted_contains(), and swap().

The documentation for this class was generated from the following files:

- Bit_Matrix.defs.hh
- Bit_Matrix.inlines.hh
- Bit_Matrix.cc

11.8 Parma_Polyhedra_Library::Bit_Row Class Reference

A row in a matrix of bits.

`#include <Bit_Row.defs.hh>`

Public Member Functions

- Bit_Row ()
  Default constructor.
- Bit_Row (const Bit_Row &y)
  Copy constructor.
- Bit_Row (const Bit_Row &y, const Bit_Row &z)
  Set-union constructor.
- ~Bit_Row ()
  Destructor.
- Bit_Row & operator= (const Bit_Row &y)
  Assignment operator.
- void swap (Bit_Row &y)
  Swaps `*this` with `y`.
- bool operator[ ] (unsigned long k) const
  Returns the truth value corresponding to the bit in position `k`.
- void set (unsigned long k)
  Sets the bit in position `k`.
- void set_until (unsigned long k)

Sets bits up to position `k` (excluded).

- void clear (unsigned long k)

  Clears the bit in position `k`.

- void clear_from (unsigned long k)

  Clears bits from position `k` (included) onward.

- void clear ()

  Clears all the bits of the row.

- unsigned long first () const

  Returns the index of the first set bit or ULONG_MAX if no bit is set.

- unsigned long next (unsigned long position) const

  Returns the index of the first set bit after `position` or ULONG_MAX if no bit after `position` is set.

- unsigned long last () const

  Returns the index of the last set bit or ULONG_MAX if no bit is set.

- unsigned long prev (unsigned long position) const

  Returns the index of the first set bit before `position` or ULONG_MAX if no bits before `position` is set.

- unsigned long count_ones () const

  Returns the number of set bits in the row.

- bool empty () const

  Returns `true` if no bit is set in the row.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

Private Member Functions

- void union_helper (const Bit_Row &x, const Bit_Row &y)

  Assigns to `*this` the union of `y` and `z`.

Private Attributes

- mpz_t vec

  Bit-vector representing the row.

Friends

- int compare (const Bit_Row &x, const Bit_Row &y)

  The basic comparison function.

- bool operator== (const Bit_Row &x, const Bit_Row &y)

  Returns `true` *if and only if* `x` *and* `y` *are equal.*

- bool operator!= (const Bit_Row &x, const Bit_Row &y)

  Returns `true` *if and only if* `x` *and* `y` *are not equal.*

- bool subset_or_equal (const Bit_Row &x, const Bit_Row &y)

  Set-theoretic inclusion test.

- bool subset_or_equal (const Bit_Row &x, const Bit_Row &y, bool &strict_subset)

  Set-theoretic inclusion test: sets `strict_subset` *to a Boolean indicating whether the inclusion is strict or not.*

- bool strict_subset (const Bit_Row &x, const Bit_Row &y)

  Set-theoretic strict inclusion test.

- void set_union (const Bit_Row &x, const Bit_Row &y, Bit_Row &z)

  Set-theoretic union.

- void set_intersection (const Bit_Row &x, const Bit_Row &y, Bit_Row &z)

  Set-theoretic intersection.

- void set_difference (const Bit_Row &x, const Bit_Row &y, Bit_Row &z)

  Set-theoretic difference.

Related Functions

(Note that these are not member functions.)

- void swap (Parma_Polyhedra_Library::Bit_Row &x, Parma_Polyhedra_Library::Bit_Row &y)

  Specializes `std::swap`.

- void iter_swap (std::vector< Parma_Polyhedra_Library::Bit_Row >::iterator x, std::vector< Parma_Polyhedra_Library::Bit_Row >::iterator y)

  Specializes `std::iter_swap`.

11.8.1 Detailed Description

A row in a matrix of bits.

Definition at line 112 of file Bit_Row.defs.hh.

11.8.2 Constructor & Destructor Documentation

11.8.2.1 Parma_Polyhedra_Library::Bit_Row::Bit_Row () [inline]

Default constructor.

Definition at line 42 of file Bit_Row.inlines.hh.

References vec.

11.8.2.2 Parma_Polyhedra_Library::Bit_Row::Bit_Row (const Bit_Row & *y*) [inline]

Copy constructor.

Definition at line 47 of file Bit_Row.inlines.hh.

References vec.

11.8.2.3 Parma_Polyhedra_Library::Bit_Row::Bit_Row (const Bit_Row & *y*, const Bit_Row & *z*) [inline]

Set-union constructor.

Constructs an object containing the set-union of `y` and `z`.

Definition at line 52 of file Bit_Row.inlines.hh.

References PPL_BITS_PER_GMP_LIMB, union_helper(), and vec.

11.8.2.4 Parma_Polyhedra_Library::Bit_Row::~Bit_Row () [inline]

Destructor.

Definition at line 72 of file Bit_Row.inlines.hh.

References vec.

11.8.3 Member Function Documentation

11.8.3.1 void Parma_Polyhedra_Library::Bit_Row::clear () [inline]

Clears all the bits of the row.

Definition at line 115 of file Bit_Row.inlines.hh.

References vec.

11.8.3.2 void Parma_Polyhedra_Library::Bit_Row::clear (unsigned long *k*) [inline]

Clears the bit in position `k`.

Definition at line 88 of file Bit_Row.inlines.hh.

References vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.8.3.3 void Parma_Polyhedra_Library::Bit_Row::clear_from (unsigned long *k*) [inline]

Clears bits from position `k` (included) onward.

Definition at line 93 of file Bit_Row.inlines.hh.

References vec.

11.8.3.4 unsigned long Parma_Polyhedra_Library::Bit_Row::count_ones () const [inline]

Returns the number of set bits in the row.

Definition at line 98 of file Bit_Row.inlines.hh.

References vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion().

11.8.3.5 bool Parma_Polyhedra_Library::Bit_Row::empty () const [inline]

Returns `true` if no bit is set in the row.

Definition at line 105 of file Bit_Row.inlines.hh.

References vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), and Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign().

11.8.3.6 memory_size_type Parma_Polyhedra_Library::Bit_Row::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 120 of file Bit_Row.inlines.hh.

References vec.

Referenced by total_memory_in_bytes().

11.8.3.7 unsigned long Parma_Polyhedra_Library::Bit_Row::first () const

Returns the index of the first set bit or ULONG_MAX if no bit is set.

Definition at line 33 of file Bit_Row.cc.

References Parma_Polyhedra_Library::Implementation::first_one(), PPL_BITS_PER_GMP_LIMB, and vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact().

11.8.3.8 unsigned long Parma_Polyhedra_Library::Bit_Row::last () const

Returns the index of the last set bit or ULONG_MAX if no bit is set.

Definition at line 82 of file Bit_Row.cc.

References Parma_Polyhedra_Library::Implementation::last_one(), PPL_BITS_PER_GMP_LIMB, and vec.

Referenced by Parma_Polyhedra_Library::Bit_Matrix::OK().

11.8.3.9 unsigned long Parma_Polyhedra_Library::Bit_Row::next (unsigned long *position*) const

Returns the index of the first set bit after `position` or ULONG_MAX if no bit after `position` is set.

Definition at line 47 of file Bit_Row.cc.

References Parma_Polyhedra_Library::Implementation::first_one(), PPL_BITS_PER_GMP_LIMB, and vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact().

11.8.3.10 bool Parma_Polyhedra_Library::Bit_Row::OK () const

Checks if all the invariants are satisfied.

Definition at line 302 of file Bit_Row.cc.

References vec.

Referenced by Parma_Polyhedra_Library::Bit_Matrix::OK().

11.8.3.11 Bit_Row & Parma_Polyhedra_Library::Bit_Row::operator= (const Bit_Row & *y*) `[inline]`

Assignment operator.

Definition at line 77 of file Bit_Row.inlines.hh.

References vec.

11.8.3.12 bool Parma_Polyhedra_Library::Bit_Row::operator[ ] (unsigned long *k*) const

Returns the truth value corresponding to the bit in position `k`.

Definition at line 135 of file Bit_Row.cc.

References vec.

11.8.3.13 unsigned long Parma_Polyhedra_Library::Bit_Row::prev (unsigned long *position*) const

Returns the index of the first set bit before `position` or ULONG_MAX if no bits before `position` is set.

Definition at line 95 of file Bit_Row.cc.

References Parma_Polyhedra_Library::Implementation::last_one(), PPL_BITS_PER_GMP_LIMB, and vec.

11.8.3.14 void Parma_Polyhedra_Library::Bit_Row::set (unsigned long *k*) [inline]

Sets the bit in position `k`.

Definition at line 83 of file Bit_Row.inlines.hh.

References vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_reduction_assign(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators().

11.8.3.15 void Parma_Polyhedra_Library::Bit_Row::set_until (unsigned long *k*)

Sets bits up to position `k` (excluded).

Definition at line 148 of file Bit_Row.cc.

References vec.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), and Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact().

11.8.3.16 void Parma_Polyhedra_Library::Bit_Row::swap (Bit_Row & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 110 of file Bit_Row.inlines.hh.

References vec.

Referenced by swap().

11.8.3.17 memory_size_type Parma_Polyhedra_Library::Bit_Row::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 125 of file Bit_Row.inlines.hh.

References external_memory_in_bytes().

11.8.3.18 void Parma_Polyhedra_Library::Bit_Row::union_helper (const Bit_Row & *x*, const Bit_Row & *y*) [private]

Assigns to `*this` the union of `y` and `z`.

The size of `y` must be be less than or equal to the size of `z`. Upon entry, `vec` must have allocated enough space to contain the result.

Definition at line 311 of file Bit_Row.cc.

References vec.

Referenced by Bit_Row().

11.8.4 Friends And Related Function Documentation

11.8.4.1 int compare (const Bit_Row & *x*, const Bit_Row & *y*) [friend]

The basic comparison function.

Compares `x` with `y` starting from the least significant bits. The ordering is total and has the following property: if `x` and `y` are two rows seen as sets of naturals, if `x` is a strict subset of `y`, then `x` comes before `y`.

Returns

- -1 if `x` comes before `y` in the ordering;
- 0 if `x` and `y` are equal;
- 1 if `x` comes after `y` in the ordering.

Definition at line 156 of file Bit_Row.cc.

11.8.4.2 void iter_swap (std::vector< Parma_Polyhedra_Library::Bit_Row >::iterator *x*, std::vector< Parma_Polyhedra_Library::Bit_Row >::iterator *y*) [related]

Specializes `std::iter_swap`.

Definition at line 336 of file Bit_Row.inlines.hh.

References Parma_Polyhedra_Library::swap().

11.8.4.3 bool operator!= (const Bit_Row & *x*, const Bit_Row & *y*) [friend]

Returns `true` if and only if `x` and `y` are not equal.

Definition at line 289 of file Bit_Row.cc.

11.8.4.4 bool operator== (const Bit_Row & *x*, const Bit_Row & *y*) [friend]

Returns `true` if and only if `x` and `y` are equal.

Definition at line 275 of file Bit_Row.cc.

11.8.4.5 void set_difference (const Bit_Row & *x*, const Bit_Row & *y*, Bit_Row & *z*) [friend]

Set-theoretic difference.

Definition at line 158 of file Bit_Row.inlines.hh.

11.8.4.6 void set_intersection (const Bit_Row & *x*, const Bit_Row & *y*, Bit_Row & *z*) [friend]

Set-theoretic intersection.

Definition at line 152 of file Bit_Row.inlines.hh.

11.8.4.7 void set_union (const Bit_Row & *x*, const Bit_Row & *y*, Bit_Row & *z*) [friend]

Set-theoretic union.

Definition at line 131 of file Bit_Row.inlines.hh.

11.8.4.8 bool strict_subset (const Bit_Row & *x*, const Bit_Row & *y*) [friend]

Set-theoretic strict inclusion test.

Definition at line 249 of file Bit_Row.cc.

11.8.4.9 bool subset_or_equal (const Bit_Row & *x*, const Bit_Row & *y*, bool & *strict_subset*) [friend]

Set-theoretic inclusion test: sets `strict_subset` to a Boolean indicating whether the inclusion is strict or not.

Definition at line 204 of file Bit_Row.cc.

11.8.4.10 bool subset_or_equal (const Bit_Row & *x*, const Bit_Row & *y*) [friend]

Set-theoretic inclusion test.

Definition at line 183 of file Bit_Row.cc.

11.8.4.11 void swap (Parma_Polyhedra_Library::Bit_Row & *x*, Parma_Polyhedra_Library::Bit_Row & *y*) [related]

Specializes `std::swap`.

Definition at line 329 of file Bit_Row.inlines.hh.

References swap().

11.8.5 Member Data Documentation

11.8.5.1 mpz_t Parma_Polyhedra_Library::Bit_Row::vec [private]

Bit-vector representing the row.

Definition at line 199 of file Bit_Row.defs.hh.

Referenced by Bit_Row(), clear(), clear_from(), count_ones(), empty(), external_memory_in_bytes(), first(), last(), next(), OK(), operator=(), operator[ ](), prev(), set(), set_until(), swap(), union_helper(), and ~Bit_Row().

The documentation for this class was generated from the following files:

- Bit_Row.defs.hh
- Bit_Row.inlines.hh
- Bit_Row.cc

11.9 Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than Struct Reference

Ordering predicate (used when implementing the sort algorithm).

Public Member Functions

- bool operator() (const Bit_Row &x, const Bit_Row &y) const

11.9.1 Detailed Description

Ordering predicate (used when implementing the sort algorithm).

Definition at line 147 of file Bit_Matrix.defs.hh.

11.9.2 Member Function Documentation

11.9.2.1 bool Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than::operator() (const Bit_Row & *x*, const Bit_Row & *y*) const [inline]

Definition at line 121 of file Bit_Matrix.inlines.hh.

The documentation for this struct was generated from the following files:

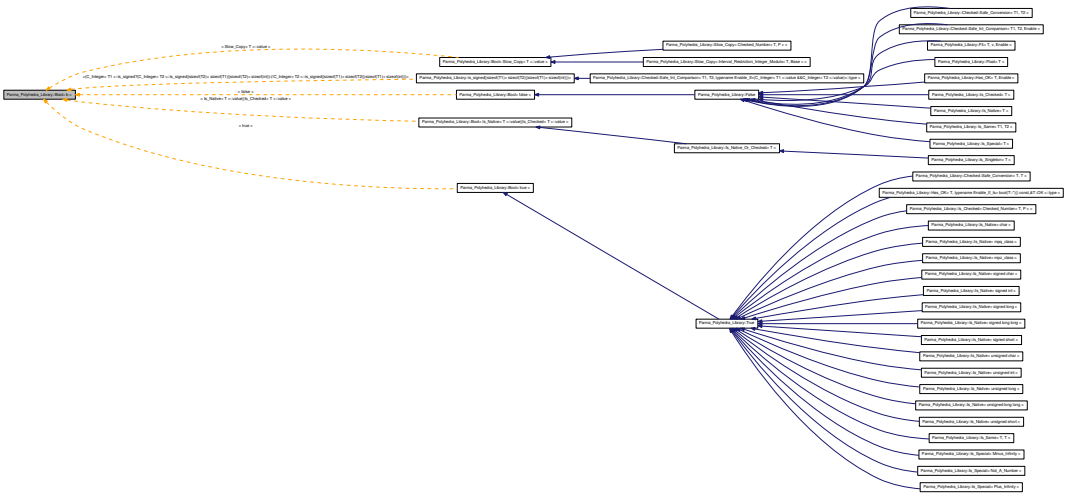
- Bit_Matrix.defs.hh
- Bit_Matrix.inlines.hh

11.10 Parma_Polyhedra_Library::Bool< b > Struct Template Reference

A class holding a constant called `value` that evaluates to `b`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Bool< b >:



Public Types

- enum anonymous_enum { value = b }

11.10.1 Detailed Description

template<bool b> struct Parma_Polyhedra_Library::Bool< b >

A class holding a constant called `value` that evaluates to `b`.

Definition at line 144 of file meta_programming.hh.

11.10.2 Member Enumeration Documentation

11.10.2.1 template<bool b> enum Parma_Polyhedra_Library::Bool::anonymous_enum

Enumerator:

value

Definition at line 145 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.11 Parma_Polyhedra_Library::Boundary_Value< T, Enable > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef T type

11.11.1 Detailed Description

template<typename T, typename Enable = void> struct Parma_Polyhedra_Library::Boundary_Value< T, Enable >

Definition at line 39 of file Interval_Restriction.defs.hh.

11.11.2 Member Typedef Documentation

11.11.2.1 template<typename T, typename Enable = void> typedef T Parma_Polyhedra_Library::Boundary_Value< T, Enable >::type

Definition at line 40 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.12 Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef T::boundary_type type

11.12.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type >

Definition at line 44 of file Interval_Restriction.defs.hh.

11.12.2 Member Typedef Documentation

11.12.2.1 template<typename T > typedef T::boundary_type Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type >::type

Definition at line 45 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

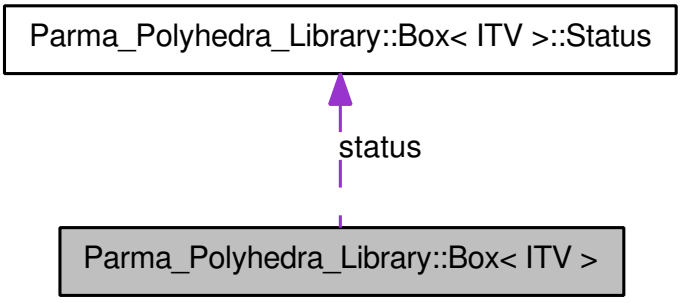
- Interval_Restriction.defs.hh

11.13 Parma_Polyhedra_Library::Box< ITV > Class Template Reference

A not necessarily closed, iso-oriented hyperrectangle.

`#include <Box.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Box< ITV >:



Classes

- class Status

Public Types

- typedef ITV interval_type

  The type of intervals used to implement the box.

Public Member Functions

- const ITV & get_interval (Variable var) const

  Returns a reference the interval that bounds `var`.

- void set_interval (Variable var, const ITV &i)

  Sets to `i` *the interval that bounds* `var`.

- bool get_lower_bound (dimension_type k, bool &closed, Coefficient &n, Coefficient &d) const

  If the `k`*-th space dimension is unbounded below, returns* `false`*. Otherwise returns* `true` *and set* `closed`*,* `n` *and* `d` *accordingly.*

- bool get_upper_bound (dimension_type k, bool &closed, Coefficient &n, Coefficient &d) const

  If the `k`*-th space dimension is unbounded above, returns* `false`*. Otherwise returns* `true` *and set* `closed`*,* `n` *and* `d` *accordingly.*

- Constraint_System constraints () const

  Returns a system of constraints defining `*this`.

- Constraint_System minimized_constraints () const

  Returns a minimized system of constraints defining `*this`.

- Congruence_System congruences () const

  Returns a system of congruences approximating `*this`.

- Congruence_System minimized_congruences () const

  Returns a minimized system of congruences approximating `*this`.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- void set_empty ()

  Causes the box to become empty, i.e., to represent the empty set.

Constructors, Assignment, Swap and Destructor

- Box (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

  Builds a universe or empty box of the specified space dimension.

- Box (const Box &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- template<typename Other_ITV >
  Box (const Box< Other_ITV > &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a conservative, upward approximation of `y`.

- Box (const Constraint_System &cs)

  Builds a box from the system of constraints `cs`.

- Box (const Constraint_System &cs, Recycle_Input dummy)

  Builds a box recycling a system of constraints `cs`.

- Box (const Generator_System &gs)

  Builds a box from the system of generators `gs`.

- Box (const Generator_System &gs, Recycle_Input dummy)

  Builds a box recycling the system of generators `gs`.

- Box (const Congruence_System &cgs)
- Box (const Congruence_System &cgs, Recycle_Input dummy)
- template<typename T >
  Box (const BD_Shape< T > &bds, Complexity_Class complexity=POLYNOMIAL_COMPLEXITY)

  Builds a box containing the BDS `bds`.

- template<typename T >
  Box (const Octagonal_Shape< T > &oct, Complexity_Class complexity=POLYNOMIAL_COMPLEXITY)

  Builds a box containing the octagonal shape `oct`.

- Box (const Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a box containing the polyhedron `ph`.

- Box (const Grid &ph, Complexity_Class complexity=POLYNOMIAL_COMPLEXITY)

  Builds a box containing the grid `gr`.

- template<typename D1 , typename D2 , typename R >
  Box (const Partially_Reduced_Product< D1, D2, R > &dp, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a box containing the partially reduced product `dp`.

- Box & operator= (const Box &y)
  The assignment operator (`*this` *and* `y` *can be dimension-incompatible).*

- void swap (Box &y)
  Swaps `*this` *with* `y` *(*`*this` *and* `y` *can be dimension-incompatible).*

Member Functions that Do Not Modify the Box

- dimension_type space_dimension () const
  Returns the dimension of the vector space enclosing `*this`.

- dimension_type affine_dimension () const
  Returns 0*, if* `*this` *is empty; otherwise, returns the affine dimension of* `*this`.

- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty box.*

- bool is_universe () const
  Returns `true` *if and only if* `*this` *is a universe box.*

- bool is_topologically_closed () const
  Returns `true` *if and only if* `*this` *is a topologically closed subset of the vector space.*

- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*

- bool is_bounded () const
  Returns `true` *if and only if* `*this` *is a bounded box.*

- bool contains_integer_point () const
  Returns `true` *if and only if* `*this` *contains at least one integer point.*

- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.

- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between `*this` *and the constraint* `c`.

- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between `*this` *and the congruence* `cg`.

- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between `*this` *and the generator* `g`.

- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from above in* `*this`.

- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from below in* `*this`.

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &g) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &g) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool frequency (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const

  Returns `true` *if and only if there exist a unique value* `val` *such that* `*this` *saturates the equality* `expr = val`*.*

- bool contains (const Box &y) const

  Returns `true` *if and only if* `*this` *contains* `y`*.*

- bool strictly_contains (const Box &y) const

  Returns `true` *if and only if* `*this` *strictly contains* `y`*.*

- bool is_disjoint_from (const Box &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*

- bool OK () const

  Returns `true` *if and only if* `*this` *satisfies all its invariants.*

Space-Dimension Preserving Member Functions that May Modify the Box

- void add_constraint (const Constraint &c)

  Adds a copy of constraint `c` *to the system of constraints defining* `*this`*.*

- void add_constraints (const Constraint_System &cs)

  Adds the constraints in `cs` *to the system of constraints defining* `*this`*.*

- void add_recycled_constraints (Constraint_System &cs)

  Adds the constraints in `cs` *to the system of constraints defining* `*this`*.*

- void add_congruence (const Congruence &cg)

  Adds to `*this` *a constraint equivalent to the congruence* `cg`*.*

- void add_congruences (const Congruence_System &cgs)

  Adds to `*this` *constraints equivalent to the congruences in* `cgs`*.*

- void add_recycled_congruences (Congruence_System &cgs)
  Adds to `*this` *constraints equivalent to the congruences in* `cgs`.

- void refine_with_constraint (const Constraint &c)
  Use the constraint `c` *to refine* `*this`.

- void refine_with_constraints (const Constraint_System &cs)
  Use the constraints in `cs` *to refine* `*this`.

- void refine_with_congruence (const Congruence &cg)
  Use the congruence `cg` *to refine* `*this`.

- void refine_with_congruences (const Congruence_System &cgs)
  Use the congruences in `cgs` *to refine* `*this`.

- void propagate_constraint (const Constraint &c)
  Use the constraint `c` *for constraint propagation on* `*this`.

- void propagate_constraints (const Constraint_System &cs, dimension_type max_iterations=0)
  Use the constraints in `cs` *for constraint propagagion on* `*this`.

- void unconstrain (Variable var)
  Computes the cylindrification of `*this` *with respect to space dimension* `var`*, assigning the result to* `*this`.

- void unconstrain (const Variables_Set &vars)
  Computes the cylindrification of `*this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `*this`.

- void intersection_assign (const Box &y)
  Assigns to `*this` *the intersection of* `*this` *and* `y`.

- void upper_bound_assign (const Box &y)
  Assigns to `*this` *the smallest box containing the union of* `*this` *and* `y`.

- bool upper_bound_assign_if_exact (const Box &y)
  If the upper bound of `*this` *and* `y` *is exact, it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- void difference_assign (const Box &y)
  Assigns to `*this` *the difference of* `*this` *and* `y`.

- bool simplify_using_context_assign (const Box &y)
  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`*. If* `false` *is returned, then the intersection is empty.*

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())
  Assigns to `*this` *the affine image of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())
  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void time_elapse_assign (const Box &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`.

- void topological_closure_assign ()

  Assigns to `*this` *its topological closure.*

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)

  Wraps the specified dimensions of the vector space.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates for the space dimensions corresponding to* `vars`.

- template<typename T >
  Enable_If< Is_Same< T, Box >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type CC76_widening_assign (const T &y, unsigned *tp=0)

  Assigns to `*this` *the result of computing the CC76-widening between* `*this` *and* `y`.

- template<typename T , typename Iterator >
  Enable_If< Is_Same< T, Box >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type CC76_widening_assign (const T &y, Iterator first, Iterator last)
  Assigns to `*this` the result of computing the CC76-widening between `*this` and `y`.

- void widening_assign (const Box &y, unsigned *tp=0)
  Same as CC76_widening_assign(y, tp).

- void limited_CC76_extrapolation_assign (const Box &y, const Constraint_System &cs, unsigned *tp=0)
  Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

- template<typename T >
  Enable_If< Is_Same< T, Box >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type CC76_narrowing_assign (const T &y)
  Assigns to `*this` the result of restoring in `y` the constraints of `*this` that were lost by CC76-extrapolation applications.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)
  Adds `m` new dimensions and embeds the old box into the new space.

- void add_space_dimensions_and_project (dimension_type m)
  Adds `m` new dimensions to the box and does not embed it in the new vector space.

- void concatenate_assign (const Box &y)
  Seeing a box as a set of tuples (its points), assigns to `*this` all the tuples that can be obtained by concatenating, in the order given, a tuple of `*this` with a tuple of `y`.

- void remove_space_dimensions (const Variables_Set &vars)
  Removes all the specified dimensions.

- void remove_higher_space_dimensions (dimension_type new_dimension)
  Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)
  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)
  Creates `m` copies of the space dimension corresponding to `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` into `dest`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension that a Box can handle.

- static bool can_recycle_constraint_systems ()

  Returns false indicating that this domain does not recycle constraints.

- static bool can_recycle_congruence_systems ()

  Returns false indicating that this domain does not recycle congruences.

Private Types

- typedef std::vector< ITV > Sequence

  The type of sequence used to implement the box.

- typedef ITV Tmp_Interval_Type

  The type of intervals used by inner computations when trying to limit the cumulative effect of approximation errors.

Private Member Functions

- bool marked_empty () const

  Returns `true` *if and only if the box is known to be empty.*

- void set_nonempty ()

  Marks `*this` *as definitely not empty.*

- void set_empty_up_to_date ()

  Asserts the validity of the empty flag of `*this`*.*

- void reset_empty_up_to_date ()

  Invalidates empty flag of `*this`*.*

- bool check_empty () const

  Checks the hard way whether `*this` *is an empty box: returns* `true` *if and only if it is so.*

- const ITV & operator[ ] (dimension_type k) const

  Returns a reference the interval that bounds the box on the `k`*-th space dimension.*

- void add_interval_constraint_no_check (dimension_type var_id, Constraint::Type type, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den)

  WRITE ME.

- void add_constraint_no_check (const Constraint &c)

  WRITE ME.

- void add_constraints_no_check (const Constraint_System &cs)

  WRITE ME.

- void add_congruence_no_check (const Congruence &cg)

  WRITE ME.

- void add_congruences_no_check (const Congruence_System &cgs)

  WRITE ME.

- void refine_no_check (const Constraint &c)

  Uses the constraint `c` *to refine* `*this`.

- void refine_no_check (const Constraint_System &cs)

  Uses the constraints in `cs` *to refine* `*this`.

- void refine_no_check (const Congruence &cg)

  Uses the congruence `cg` *to refine* `*this`.

- void refine_no_check (const Congruence_System &cgs)

  Uses the congruences in `cgs` *to refine* `*this`.

- void propagate_constraint_no_check (const Constraint &c)

  Propagates the constraint `c` *to refine* `*this`.

- void propagate_constraints_no_check (const Constraint_System &cs, dimension_type max_iterations)

  Propagates the constraints in `cs` *to refine* `*this`.

- bool bounds (const Linear_Expression &expr, bool from_above) const

  Checks if and how `expr` *is bounded in* `*this`.

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included, Generator &g) const

  Maximizes or minimizes `expr` *subject to* `*this`.

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included) const

  Maximizes or minimizes `expr` *subject to* `*this`.

- void get_limiting_box (const Constraint_System &cs, Box &limiting_box) const

  Adds to `limiting_box` *the interval constraints in* `cs` *that are satisfied by* `*this`.

Static Private Member Functions

- static I_Result refine_interval_no_check (ITV &itv, Constraint::Type type, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den)

  WRITE ME.

Private Attributes

- Sequence seq

  A sequence of intervals, one for each dimension of the vector space.

- Status status

The status flags to keep track of the internal state.

Friends

- class Parma_Polyhedra_Library::Box
- bool operator== (const Box< ITV > &x, const Box< ITV > &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same box.*

- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Box< ITV > &box)
- template<typename Specialization , typename Temp , typename To , typename I >
  bool Parma_Polyhedra_Library::l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< I > &x, const Box< I > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

Related Functions

(Note that these are not member functions.)

- template<typename ITV >
  bool operator!= (const Box< ITV > &x, const Box< ITV > &y)

  Returns `true` *if and only if* `x` *and* `y` *aren't the same box.*

- template<typename ITV >
  std::ostream & operator<< (std::ostream &s, const Box< ITV > &box)

  Output operator.

- template<typename To , typename ITV >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename ITV >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename To , typename ITV >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename ITV >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename To , typename ITV >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir)

  Computes the L_∞ distance between x and y.

- template<typename Temp , typename To , typename ITV >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ distance between x and y.

- template<typename Specialization , typename Temp , typename To , typename ITV >
  bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Box< ITV > &x, const Box< ITV > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)
- bool extract_interval_constraint (const Constraint &c, dimension_type c_space_dim, dimension_type &c_num_vars, dimension_type &c_only_var)

  Decodes the constraint c as an interval constraint.

Exception Throwers

- void throw_dimension_incompatible (const char *method, const Box &x) const
- void throw_dimension_incompatible (const char *method, dimension_type required_dim) const
- void throw_dimension_incompatible (const char *method, const Constraint &c) const
- void throw_dimension_incompatible (const char *method, const Congruence &cg) const
- void throw_dimension_incompatible (const char *method, const Constraint_System &cs) const
- void throw_dimension_incompatible (const char *method, const Congruence_System &cgs) const
- void throw_dimension_incompatible (const char *method, const Generator &g) const
- void throw_dimension_incompatible (const char *method, const char *name_row, const Linear_Expression &y) const
- static void throw_space_dimension_overflow (const char *method, const char *reason)
- static void throw_constraint_incompatible (const char *method)
- static void throw_expression_too_complex (const char *method, const Linear_Expression &e)
- static void throw_generic (const char *method, const char *reason)

11.13.1 Detailed Description

template<typename ITV> class Parma_Polyhedra_Library::Box< ITV >

A not necessarily closed, iso-oriented hyperrectangle. A Box object represents the smash product of n not necessarily closed and possibly unbounded intervals represented by objects of class `ITV`, where n is the space dimension of the box.

An *interval constraint* (resp., *interval congruence*) is a syntactic constraint (resp., congruence) that only mentions a single space dimension.

The Box domain *optimally supports*:

- tautological and inconsistent constraints and congruences;
- the interval constraints that are optimally supported by the template argument class `ITV`;
- the interval congruences that are optimally supported by the template argument class `ITV`.

Depending on the method, using a constraint or congruence that is not optimally supported by the domain will either raise an exception or result in a (possibly non-optimal) upward approximation.

The user interface for the Box domain is meant to be as similar as possible to the one developed for the polyhedron class C_Polyhedron.

Definition at line 292 of file Box.defs.hh.

11.13.2 Member Typedef Documentation

11.13.2.1 template<typename ITV> typedef ITV Parma_Polyhedra_Library::Box< ITV >::interval_type

The type of intervals used to implement the box.

Definition at line 295 of file Box.defs.hh.

11.13.2.2 template<typename ITV> typedef std::vector<ITV> Parma_Polyhedra_Library::Box< ITV >::Sequence [private]

The type of sequence used to implement the box.

Definition at line 1693 of file Box.defs.hh.

11.13.2.3 template<typename ITV> typedef ITV Parma_Polyhedra_Library::Box< ITV >::Tmp_Interval_Type [private]

The type of intervals used by inner computations when trying to limit the cumulative effect of approximation errors.

Definition at line 1699 of file Box.defs.hh.

11.13.3 Constructor & Destructor Documentation

11.13.3.1 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds a universe or empty box of the specified space dimension.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the box;

kind Specifies whether the universe or the empty box has to be built.

Definition at line 48 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.3.2 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Box< ITV > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 70 of file Box.inlines.hh.

11.13.3.3 template<typename ITV > template<typename Other_ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Box< Other_ITV > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a conservative, upward approximation of `y`.

The complexity argument is ignored.

Definition at line 103 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.3.4 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Constraint_System & *cs*) [inline, explicit]

Builds a box from the system of constraints `cs`.

The box inherits the space dimension of `cs`.

Parameters

cs A system of constraints: constraints that are not interval constraints are ignored (even though they may have contributed to the space dimension).

Definition at line 70 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.3.5 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Constraint_System & *cs*, Recycle_Input *dummy*) [inline]

Builds a box recycling a system of constraints `cs`.

The box inherits the space dimension of cs.

Parameters

cs A system of constraints: constraints that are not interval constraints are ignored (even though they may have contributed to the space dimension).

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Definition at line 92 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::swap().

11.13.3.6 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Generator_System & *gs*) [inline, explicit]

Builds a box from the system of generators gs.

Builds the smallest box containing the polyhedron defined by gs. The box inherits the space dimension of gs.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 119 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Generator::type(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.3.7 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Generator_System & *gs*, Recycle_Input *dummy*) [inline]

Builds a box recycling the system of generators gs.

Builds the smallest box containing the polyhedron defined by gs. The box inherits the space dimension of gs.

Parameters

gs The generator system describing the polyhedron to be approximated.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 100 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::swap().

11.13.3.8 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Congruence_System & *cgs*) [inline, explicit]

Builds the smallest box containing the grid defined by a system of congruences `cgs`. The box inherits the space dimension of `cgs`.

Parameters

cgs A system of congruences: congruences that are not non-relational equality constraints are ignored (though they may have contributed to the space dimension).

Definition at line 86 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.3.9 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Congruence_System & *cgs*, Recycle_Input *dummy*) [inline]

Builds the smallest box containing the grid defined by a system of congruences `cgs`, recycling `cgs`. The box inherits the space dimension of `cgs`.

Parameters

cgs A system of congruences: congruences that are not non-relational equality constraints are ignored (though they will contribute to the space dimension).

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Definition at line 108 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::swap().

11.13.3.10 template<typename ITV > template<typename T > Parma_Polyhedra_Library::Box< ITV >::Box (const BD_Shape< T > & *bds*, Complexity_Class *complexity* = POLYNOMIAL_COMPLEXITY) [inline, explicit]

Builds a box containing the BDS `bds`.

Builds the smallest box containing `bds` using a polynomial algorithm. The `complexity` argument is ignored.

Definition at line 224 of file Box.templates.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::dbm, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::BD_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::BD_Shape< T >::shortest_path_closure_assign(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.3.11 template<typename ITV > template<typename T > Parma_Polyhedra_Library::Box< ITV >::Box (const Octagonal_Shape< T > & *oct*, Complexity_Class *complexity* = POLYNOMIAL_COMPLEXITY) [inline, explicit]

Builds a box containing the octagonal shape `oct`.

Builds the smallest box containing `oct` using a polynomial algorithm. The `complexity` argument is ignored.

Definition at line 276 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.13.3.12 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a box containing the polyhedron `ph`.

Builds a box containing `ph` using algorithms whose complexity does not exceed the one specified by `complexity`. If `complexity` is `ANY_COMPLEXITY`, then the built box is the smallest one containing `ph`.

Definition at line 329 of file Box.templates.hh.

References Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::Polyhedron::has_pending_constraints(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::is_canonical(), Parma_Polyhedra_Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::MIP_Problem::is_satisfiable(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, Parma_Polyhedra_Library::MIP_Problem::optimizing_point(), Parma_Polyhedra_Library::POLYNOMIAL_COMPLEXITY, Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::MIP_Problem::set_objective_function(), Parma_Polyhedra_Library::MIP_Problem::set_optimization_mode(), Parma_Polyhedra_Library::SIMPLEX_COMPLEXITY, Parma_Polyhedra_Library::Polyhedron::simplified_constraints(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Polyhedron::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::swap(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.3.13 template<typename ITV > Parma_Polyhedra_Library::Box< ITV >::Box (const Grid & *ph*, Complexity_Class *complexity* = POLYNOMIAL_COMPLEXITY) [inline, explicit]

Builds a box containing the grid `gr`.

Builds the smallest box containing `gr` using a polynomial algorithm. The `complexity` argument is ignored.

Definition at line 440 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Grid_Generator_System::empty(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::Grid::gen_sys, Parma_Polyhedra_Library::Grid::generators_are_up_to_date(), Parma_Polyhedra_Library::Grid::marked_empty(), Parma_Polyhedra_Library::Grid::maximize(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::Grid::space_dimension(), Parma_Polyhedra_Library::UNIVERSE, and Parma_Polyhedra_Library::Grid::update_generators().

11.13.3.14 template<typename ITV > template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Box< ITV >::Box (const Partially_Reduced_Product< D1, D2, R > & *dp*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a box containing the partially reduced product `dp`.

Builds a box containing `ph` using algorithms whose complexity does not exceed the one specified by `complexity`.

Definition at line 495 of file Box.templates.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain1(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain2(), Parma_Polyhedra_Library::Box< ITV >::max_space_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::swap(), and Parma_Polyhedra_Library::Box< ITV >::throw_space_dimension_overflow().

11.13.4 Member Function Documentation

11.13.4.1 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_congruence (const Congruence & *cg*) [inline]

Adds to `*this` a constraint equivalent to the congruence `cg`.

Parameters

cg The congruence to be added.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible, or `cg` is not optimally supported by the box domain.

Definition at line 327 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.2 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check (const Congruence & *cg*) [inline, private]

WRITE ME.

Definition at line 2170 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_interval_constraint_no_check(), Parma_Polyhedra_Library::Congruence::coefficient(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::extract_interval_congruence(), Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_generic().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence(), and Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check().

11.13.4.3 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_congruences (const Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs The congruences to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the box domain.

Definition at line 338 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_recycled_congruences().

11.13.4.4 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check (const Congruence_System & *cgs*) [inline, private]

WRITE ME.

Definition at line 2214 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Congruence_System::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruences(), and Parma_Polyhedra_Library::Box< ITV >::Box().

11.13.4.5 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_constraint (const Constraint & *c*) [inline]

Adds a copy of constraint `c` to the system of constraints defining `*this`.

Parameters

c The constraint to be added.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible, or `c` is not optimally supported by the Box domain.

Definition at line 300 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.6 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check (const Constraint & *c*) [inline, private]

WRITE ME.

Definition at line 2120 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_interval_constraint_no_check(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_generic(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_constraint(), and Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check().

11.13.4.7 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_constraints (const Constraint_System & *cs*) [inline]

Adds the constraints in `cs` to the system of constraints defining `*this`.

Parameters

cs The constraints to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible, or `cs` contains a constraint which is not optimally supported by the box domain.

Definition at line 311 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_recycled_constraints().

11.13.4.8 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check (const Constraint_System & *cs*) [inline, private]

WRITE ME.

Definition at line 2157 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_constraints(), and Parma_Polyhedra_Library::Box< ITV >::Box().

11.13.4.9 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_interval_constraint_no_check (dimension_type *var_id*, Constraint::Type *type*, Coefficient_traits::const_reference *num*, Coefficient_traits::const_reference *den*) [inline, private]

WRITE ME.

Definition at line 421 of file Box.inlines.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), and Parma_Polyhedra_Library::Box< ITV >::refine_no_check().

11.13.4.10 template<typename T > void Parma_Polyhedra_Library::Box< T >::add_recycled_congruences (Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs The congruence system to be added to `*this`. The congruences in `cgs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the box domain.

Warning

The only assumption that can be made on `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 346 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_congruences().

11.13.4.11 template<typename T > void Parma_Polyhedra_Library::Box< T >::add_recycled_constraints (Constraint_System & *cs*) [inline]

Adds the constraints in `cs` to the system of constraints defining `*this`.

Parameters

cs The constraints to be added. They may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible, or `cs` contains a constraint which is not optimally supported by the box domain.

Warning

The only assumption that can be made on `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 321 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_constraints().

11.13.4.12 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_space_dimensions_and_embed (dimension_type *m*) [inline]

Adds `m` new dimensions and embeds the old box into the new space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new box, which is defined by a system of interval constraints in which the variables running through the new dimensions are unconstrained. For instance, when starting from the box $\mathcal{B} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the box

$$\left\{ (x, y, z)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{B} \right\}.$$

Definition at line 510 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.13 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::add_space_dimensions_and_project (dimension_type *m*) [inline]

Adds `m` new dimensions to the box and does not embed it in the new vector space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new box, which is defined by a system of bounded differences in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the box $\mathcal{B} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the box

$$\left\{ (x, y, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{B} \right\}.$$

Definition at line 522 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::OK(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.14 template<typename ITV > dimension_type Parma_Polyhedra_Library::Box< ITV >::affine_dimension () const [inline]

Returns 0, if `*this` is empty; otherwise, returns the affine dimension of `*this`.

Definition at line 1270 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.15 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the affine image of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2727 of file Box.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Box< ITV >::throw_generic().

Referenced by Parma_Polyhedra_Library::Box< ITV >::affine_preimage().

11.13.4.16 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2769 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq,

Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_generic(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.17 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 3908 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::ascii_dump(), Parma_Polyhedra_Library::Box< ITV >::Status::ascii_dump(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::status.

11.13.4.18 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Box< ITV >::ascii_dump().

11.13.4.19 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 3922 of file Box.templates.hh.

11.13.4.20 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2831 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.21 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2967 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::Variable::space_dimension().

11.13.4.22 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::bounds (const Linear_Expression & *expr*, bool *from_above*) const `[inline, private]`

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

from_above `true` if and only if the boundedness of interest is "from above".

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 552 of file Box.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::bounds_from_above(), and Parma_Polyhedra_Library::Box< ITV >::bounds_from_below().

11.13.4.23 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from above in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 181 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::bounds().

11.13.4.24 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from below in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 187 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::bounds().

11.13.4.25 template<typename T > bool Parma_Polyhedra_Library::Box< T >::can_recycle_congruence_systems () [inline, static]

Returns false indicating that this domain does not recycle congruences.

Definition at line 358 of file Box.inlines.hh.

11.13.4.26 template<typename T > bool Parma_Polyhedra_Library::Box< T >::can_recycle_constraint_systems () [inline, static]

Returns false indicating that this domain does not recycle constraints.

Definition at line 352 of file Box.inlines.hh.

11.13.4.27 template<typename ITV > template<typename T > Enable_If< Is_Same< T, Box< ITV > >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign (const T & *y*) [inline]

Assigns to `*this` the result of restoring in `y` the constraints of `*this` that were lost by CC76-extrapolation applications.

Parameters

y A Box that *must* contain `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Note

As was the case for widening operators, the argument `y` is meant to denote the value computed in the previous iteration step, whereas `*this` denotes the value computed in the current iteration step (in the *decreasing* iteration sequence). Hence, the call `x.CC76_narrowing_assign(y)` will assign to `x` the result of the computation $y \Delta x$.

Definition at line 3720 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.28 template<typename ITV > template<typename T , typename Iterator > Enable_If< Is_Same< T, Box< ITV > >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign (const T & *y*, Iterator *first*, Iterator *last*) [inline]

Assigns to `*this` the result of computing the CC76-widening between `*this` and `y`.

Parameters

y A box that *must* be contained in `*this`.

first An iterator that points to the first stop-point.

last An iterator that points one past the last stop-point.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3596 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::OK(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.29 template<typename ITV > template<typename T > Enable_If< Is_Same< T, Box< ITV > >::value &&Is_Same_Or_Derived< Interval_Base, ITV >::value, void >::type Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign (const T & *y*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the CC76-widening between `*this` and `y`.

Parameters

y A box that *must* be contained in `*this`.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3611 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), and Parma_Polyhedra_Library::Box< ITV >::contains().

Referenced by Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign(), and Parma_Polyhedra_Library::Box< ITV >::widening_assign().

11.13.4.30 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::check_empty () const [inline, private]

Checks the hard way whether `*this` is an empty box: returns `true` if and only if it is so.

Definition at line 1289 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), and Parma_Polyhedra_Library::Box< ITV >::set_nonempty().

Referenced by Parma_Polyhedra_Library::Box< ITV >::is_empty(), and Parma_Polyhedra_Library::Box< ITV >::OK().

11.13.4.31 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::concatenate_assign (const Box< ITV > & *y*) [inline]

Seeing a box as a set of tuples (its points), assigns to `*this` all the tuples that can be obtained by concatenating, in the order given, a tuple of `*this` with a tuple of `y`.

Let $B \subseteq \mathbb{R}^n$ and $D \subseteq \mathbb{R}^m$ be the boxes corresponding, on entry, to `*this` and `y`, respectively. Upon successful completion, `*this` will represent the box $R \subseteq \mathbb{R}^{n+m}$ such that

$$R \stackrel{\text{def}}{=} \left\{ (x_1, \ldots, x_n, y_1, \ldots, y_m)^{\mathrm{T}} \;\middle|\; (x_1, \ldots, x_n)^{\mathrm{T}} \in B, (y_1, \ldots, y_m)^{\mathrm{T}} \in D \right\}.$$

Another way of seeing it is as follows: first increases the space dimension of `*this` by adding `y.space_dimension()` new dimensions; then adds to the system of constraints of `*this` a renamed-apart version of the constraints of `y`.

Definition at line 1767 of file Box.templates.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::status, and Parma_Polyhedra_Library::Box< ITV >::Status::test_empty_up_to_date().

11.13.4.32 template<typename ITV > Congruence_System Parma_Polyhedra_Library::Box< ITV >::congruences () const [inline]

Returns a system of congruences approximating `*this`.

Definition at line 3848 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Congruence_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::Box< ITV >::minimized_congruences().

11.13.4.33 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::constrains (Variable *var*) const [inline]

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 1433 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::is_universe(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.34 template<typename ITV > Constraint_System Parma_Polyhedra_Library::Box< ITV >::constraints () const [inline]

Returns a system of constraints defining `*this`.

Definition at line 3767 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.13.4.35 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::contains (const Box< ITV > & *y*) const [inline]

Returns `true` if and only if `*this` contains `y`.

Exceptions

std::invalid_argument Thrown if `x` and `y` are dimension-incompatible.

Definition at line 1153 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign(), and Parma_Polyhedra_Library::Box< ITV >::strictly_contains().

11.13.4.36 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::contains_integer_point () const [inline]

Returns `true` if and only if `*this` contains at least one integer point.

Definition at line 1348 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.37 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::difference_assign (const Box< ITV > & *y*) [inline]

Assigns to `*this` the difference of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1804 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.38 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 1687 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.39 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 1671 of file Box.templates.hh.

References Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::Box< ITV >::marked_-empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), and Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points().

11.13.4.40 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension (Variable *var*, dimension_type *m*) [inline]

Creates `m` copies of the space dimension corresponding to `var`.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if `var` does not correspond to a dimension of the vector space.

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

If `*this` has space dimension n, with $n > 0$, and `var` has space dimension $k \leq n$, then the k-th space dimension is expanded to `m` new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 232 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Box< ITV >::max_-space_dimension(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Box< ITV >::throw_generic().

11.13.4.41 template<typename ITV > memory_size_type Parma_Polyhedra_Library::Box< ITV >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 3878 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::total_memory_in_bytes().

11.13.4.42 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*) [inline]

Folds the space dimensions in `vars` into `dest`.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 2085 of file Box.templates.hh.

References Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Box< ITV >::throw_generic().

11.13.4.43 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::frequency (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const [inline]

Returns `true` if and only if there exist a unique value `val` such that `*this` saturates the equality `expr = val`.

Parameters

expr The linear expression for which the frequency is needed;

freq_n If `true` is returned, the value is set to 0; Present for interface compatibility with class Grid, where the frequency can have a non-zero value;

freq_d If `true` is returned, the value is set to 1;

val_n The numerator of `val`;

val_d The denominator of `val`;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `false` is returned, then `freq_n`, `freq_d`, `val_n` and `val_d` are left untouched.

Definition at line 1359 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::normalize2(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.44 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs`.

Definition at line 3341 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.45 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 3131 of file Box.templates.hh.

References Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage().

11.13.4.46 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\text{lhs}' \bowtie \text{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs`.

Definition at line 3546 of file Box.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Box< ITV >::throw_generic().

11.13.4.47 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\text{var}' \bowtie \frac{\text{expr}}{\text{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 3196 of file Box.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::NOT_EQUAL, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.48 template<typename ITV > const ITV & Parma_Polyhedra_Library::Box< ITV >::get_interval (Variable *var*) const [inline]

Returns a reference the interval that bounds `var`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 143 of file Box.inlines.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.49 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::get_limiting_box (const Constraint_System & *cs*, Box< ITV > & *limiting_box*) const [inline, private]

Adds to `limiting_box` the interval constraints in `cs` that are satisfied by `*this`.

Definition at line 3638 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_interval_constraint_no_check(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign().

11.13.4.50 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::get_lower_bound (dimension_type *k*, bool & *closed*, Coefficient & *n*, Coefficient & *d*) const [inline]

If the k-th space dimension is unbounded below, returns false. Otherwise returns true and set closed, n and d accordingly.

Let I the interval corresponding to the k-th space dimension. If I is not bounded from below, simply return false. Otherwise, set closed, n and d as follows: closed is set to true if the the lower boundary of I is closed and is set to false otherwise; n and d are assigned the integers n and d such that the canonical fraction n/d corresponds to the greatest lower bound of I. The fraction n/d is in canonical form if and only if n and d have no common factors and d is positive, $0/1$ being the unique representation for zero.

An undefined behavior is obtained if k is greater than or equal to the space dimension of *this.

Definition at line 260 of file Box.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), and Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.13.4.51 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::get_upper_bound (dimension_type *k*, bool & *closed*, Coefficient & *n*, Coefficient & *d*) const [inline]

If the k-th space dimension is unbounded above, returns false. Otherwise returns true and set closed, n and d accordingly.

Let I the interval corresponding to the k-th space dimension. If I is not bounded from above, simply return false. Otherwise, set closed, n and d as follows: closed is set to true if the the upper boundary of I is closed and is set to false otherwise; n and d are assigned the integers n and d such that the canonical fraction n/d corresponds to the least upper bound of I.

An undefined behavior is obtained if k is greater than or equal to the space dimension of *this.

Definition at line 280 of file Box.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), and Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.13.4.52 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::intersection_assign (const Box< ITV > & *y*) [inline]

Assigns to *this the intersection of *this and y.

Exceptions

std::invalid_argument Thrown if *this and y are dimension-incompatible.

Definition at line 1711 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign().

11.13.4.53 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_bounded () const [inline]

Returns `true` if and only if `*this` is a bounded box.

Definition at line 1337 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.54 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_discrete () const [inline]

Returns `true` if and only if `*this` is discrete.

Definition at line 1326 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.55 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_disjoint_from (const Box< ITV > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are dimension-incompatible.

Definition at line 1176 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.56 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_empty () const [inline]

Returns `true` if and only if `*this` is an empty box.

Definition at line 175 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::check_empty(), and Parma_Polyhedra_Library::Box< ITV >::marked_empty().

Referenced by Parma_Polyhedra_Library::Box< ITV >::affine_dimension(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::check_empty(), Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Box< ITV >::is_bounded(), Parma_Polyhedra_Library::Box< ITV >::is_discrete(), Parma_Polyhedra_Library::Box< ITV >::is_topologically_closed(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::topological_closure_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.57 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_topologically_closed () const [inline]

Returns `true` if and only if `*this` is a topologically closed subset of the vector space.

Definition at line 1314 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.58 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::is_universe () const [inline]

Returns `true` if and only if `*this` is a universe box.

Definition at line 1303 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::constrains().

11.13.4.59 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign (const Box< ITV > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y A box that *must* be contained in `*this`.

cs The system of constraints used to improve the widened box.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible or if `cs` contains a strict inequality.

Definition at line 3668 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_constraint_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.60 template<typename ITV > template<typename Partial_Function > void Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

Parameters

pfunc The partial function specifying the destiny of each dimension.

The template type parameter Partial_Function must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty co-domain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the co-domain of the partial function.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned.

The result is undefined if `pfunc` does not encode a partial function with the properties described in the specification of the mapping operator.

Definition at line 2053 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::swap().

11.13.4.61 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::marked_empty () const [inline, private]

Returns `true` if and only if the box is known to be empty.

The return value `false` does not necessarily implies that `*this` is non-empty.

Definition at line 38 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::status, Parma_Polyhedra_Library::Box< ITV >::Status::test_empty(), and Parma_Polyhedra_Library::Box< ITV >::Status::test_empty_up_to_date().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Box< ITV >::check_empty(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Box< ITV >::contains_integer_point(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::is_disjoint_from(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::is_universe(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints(), Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign(), and Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign_if_exact().

11.13.4.62 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*) const [inline, private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `*this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in * this;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `point` are left untouched.

Definition at line 1003 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_canonical(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.63 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*, Generator & *g*) const `[inline, private]`

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `*this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in `*this`;

g When maximization or minimization succeeds, will be assigned a point or closure point where `expr` reaches the corresponding extremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `g` are left untouched.

Definition at line 1071 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::lcm_assign(), Parma_Polyhedra_Library::Generator::point(), Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::maximize(), and Parma_Polyhedra_-Library::Box< ITV >::minimize().

11.13.4.64 template<typename ITV > dimension_type Parma_Polyhedra_Library::Box< ITV >::max_space_dimension () [inline, static]

Returns the maximum space dimension that a Box can handle.

Definition at line 128 of file Box.inlines.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension().

11.13.4.65 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value;

g When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d`, `maximum` and `g` are left untouched.

Definition at line 201 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::max_min().

11.13.4.66 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 193 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::max_min().

11.13.4.67 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value;

g When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

Definition at line 217 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::max_min().

11.13.4.68 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 209 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::max_min().

11.13.4.69 template<typename ITV > Congruence_System Parma_Polyhedra_Library::Box< ITV >::minimized_congruences () const [inline]

Returns a minimized system of congruences approximating `*this`.

Definition at line 370 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::congruences().

11.13.4.70 template<typename ITV > Constraint_System Parma_Polyhedra_Library::Box< ITV >::minimized_constraints () const [inline]

Returns a minimized system of constraints defining `*this`.

Definition at line 3804 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::zero_dim_empty().

11.13.4.71 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::OK () const [inline]

Returns `true` if and only if `*this` satisfies all its invariants.

Definition at line 1245 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::check_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::status, Parma_Polyhedra_Library::Box< ITV >::Status::test_empty(), and Parma_Polyhedra_Library::Box< ITV >::Status::test_empty_up_to_date().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Box<

ITV >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Box< ITV >::add_space_-dimensions_and_project(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_-Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_-Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::expand_space_-dimension(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_-Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::map_-space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_-Library::Box< ITV >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_-Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_-Polyhedra_Library::Box< ITV >::upper_bound_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.72 template<typename ITV > Box< ITV > & Parma_Polyhedra_Library::Box< ITV >::operator= (const Box< ITV > & *y*) [inline]

The assignment operator (`*this` and `y` can be dimension-incompatible).

Definition at line 76 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::status.

11.13.4.73 template<typename ITV > const ITV & Parma_Polyhedra_Library::Box< ITV >::operator[ ] (dimension_type *k*) const [inline, private]

Returns a reference the interval that bounds the box on the `k`-th space dimension.

Definition at line 136 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.74 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.13.4.75 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::propagate_constraint (const Constraint & *c*) [inline]

Use the constraint `c` for constraint propagation on `*this`.

Parameters

c The constraint to be used for constraint propagation.

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible.

Definition at line 496 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.76 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check (const Constraint & *c*) [inline, private]

Propagates the constraint `c` to refine `*this`.

Parameters

c The constraint to be propagated.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.
This method may lead to non-termination.

For any expression e, we denote by $\uparrow e \uparrow$ (resp., $\downarrow e \downarrow$) the result of any computation that is guaranteed to yield an upper (resp., lower) approximation of e. So there exists $\epsilon \in \mathbb{R}$ with $\epsilon \geq 0$ such that $\uparrow e \uparrow = e + \epsilon$. If $\epsilon = 0$ we say that the computation of $\uparrow e \uparrow$ is *exact*; we say it is *inexact* otherwise. Similarly for $\downarrow e \downarrow$.

Consider a constraint of the general form

$$z + \sum_{i \in I} a_i x_i \bowtie 0,$$

where $z \in \mathbb{Z}$, I is a set of indices, $a_i \in \mathbb{Z}$ with $a_i \neq 0$ for each $i \in I$, and $\bowtie \in \{\geq, >, =\}$. The set I is subdivided into the disjoint sets P and N such that, for each $i \in I$, $a_i > 0$ if $i \in P$ and $a_i < 0$ if $i \in N$. Suppose that, for each $i \in P \cup N$ a variation interval $\chi_i \subseteq \mathbb{R}$ is known for x_i and that the infimum and the supremum of χ_i are denoted, respectively, by χ_i^{l} and χ_i^{u}, where $\chi_i^{\mathrm{l}}, \chi_i^{\mathrm{u}} \in \mathbb{R} \cup \{-\infty, +\infty\}$.

For each $k \in P$, we have

$$x_k \bowtie \frac{1}{a_k} \left(-z - \sum_{i \in N} a_i x_i - \sum_{\substack{i \in P \\ i \neq k}} a_i x_i \right).$$

Thus, if $\chi_i^{\mathrm{l}} \in \mathbb{R}$ for each $i \in N$ and $\chi_i^{\mathrm{u}} \in \mathbb{R}$ for each $i \in P \setminus \{k\}$, we have

$$x_k \geq \left\downarrow \frac{1}{a_k} \left(-z - \sum_{i \in N} a_i \chi_i^{\mathrm{l}} - \sum_{\substack{i \in P \\ i \neq k}} a_i \chi_i^{\mathrm{u}} \right) \right\downarrow$$

and, if $\bowtie \in \{=\}$, $\chi_i^{\mathrm{u}} \in \mathbb{R}$ for each $i \in N$ and $\chi_i^{\mathrm{l}} \in \mathbb{R}$ for each $P \setminus \{k\}$,

$$x_k \leq \left\uparrow \frac{1}{a_k}\left(-z - \sum_{i \in N} a_i \chi_i^{\mathrm{u}} - \sum_{\substack{i \in P \\ i \neq k}} a_i \chi_i^{\mathrm{l}}\right)\right\uparrow .$$

In the first inequality, the relation is strict if $\bowtie \in \{>\}$, or if $\chi_i^{\mathrm{l}} \notin \chi_i$ for some $i \in N$, or if $\chi_i^{\mathrm{u}} \notin \chi_i$ for some $i \in P \setminus \{k\}$, or if the computation is inexact. In the second inequality, the relation is strict if $\chi_i^{\mathrm{u}} \notin \chi_i$ for some $i \in N$, or if $\chi_i^{\mathrm{l}} \notin \chi_i$ for some $i \in P \setminus \{k\}$, or if the computation is inexact.

For each $k \in N$, we have

$$\frac{1}{a_k}\left(-z - \sum_{\substack{i \in N \\ i \neq k}} a_i x_i - \sum_{i \in P} a_i x_i\right) \bowtie x_k.$$

Thus, if $\chi_i^{\mathrm{l}} \in \mathbb{R}$ for each $i \in N \setminus \{k\}$ and $\chi_i^{\mathrm{u}} \in \mathbb{R}$ for each $i \in P$, we have

$$\left\uparrow \frac{1}{a_k}\left(-z - \sum_{\substack{i \in N \\ i \neq k}} a_i \chi_i^{\mathrm{l}} - \sum_{i \in P} a_i \chi_i^{\mathrm{u}}\right)\right\uparrow \geq x_k$$

and, if $\bowtie \in \{=\}$, $\chi_i^{\mathrm{u}} \in \mathbb{R}$ for each $i \in N \setminus \{k\}$ and $\chi_i^{\mathrm{l}} \in \mathbb{R}$ for each $i \in P$,

$$\left\downarrow \frac{1}{a_k}\left(-z - \sum_{\substack{i \in N \\ i \neq k}} a_i \chi_i^{\mathrm{u}} - \sum_{i \in P} a_i \chi_i^{\mathrm{l}}\right)\right\downarrow \leq x_k.$$

In the first inequality, the relation is strict if $\bowtie \in \{>\}$, or if $\chi_i^{\mathrm{u}} \notin \chi_i$ for some $i \in P$, or if $\chi_i^{\mathrm{l}} \notin \chi_i$ for some $i \in N \setminus \{k\}$, or if the computation is inexact. In the second inequality, the relation is strict if $\chi_i^{\mathrm{l}} \notin \chi_i$ for some $i \in P$, or if $\chi_i^{\mathrm{u}} \notin \chi_i$ for some $i \in N \setminus \{k\}$, or if the computation is inexact.

Definition at line 2327 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Checked::Result, Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, Parma_Polyhedra_Library::T_MAYBE, Parma_Polyhedra_Library::T_NO, Parma_Polyhedra_Library::T_YES, and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Box< ITV >::propagate_constraint(), and Parma_Polyhedra_Library::Box< ITV >::refine_no_check().

11.13.4.77 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::propagate_constraints (const Constraint_System & *cs*, dimension_type *max_iterations* = 0) [inline]

Use the constraints in `cs` for constraint propagagion on `*this`.

Parameters

cs The constraints to be used for constraint propagation.

max_iterations The maximum number of propagation steps for each constraint in `cs`. If zero (the default), the number of propagations will be unbounded, possibly resulting in an infinite loop.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Warning

This method may lead to non-termination if `max_iterations` is 0.

Definition at line 511 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.78 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check (const Constraint_System & *cs*, dimension_type *max_iterations*) [inline, private]

Propagates the constraints in `cs` to refine `*this`.

Parameters

cs The constraints to be propagated.

max_iterations The maximum number of propagation steps for each constraint in `cs`. If zero, the number of propagations will be unbounded, possibly resulting in an infinite loop.

Warning

If `cs` and `*this` are dimension-incompatible, the behavior is undefined.
This method may lead to non-termination if `max_iterations` is 0.

Definition at line 2690 of file Box.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::maybe_abandon(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Box< ITV >::propagate_constraints().

11.13.4.79 template<typename ITV > I_Result Parma_Polyhedra_Library::Box< ITV >::refine_interval_no_check (ITV & *itv*, Constraint::Type *type*, Coefficient_traits::const_reference *num*, Coefficient_traits::const_reference *den*) [inline, static, private]

WRITE ME.

Definition at line 377 of file Box.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, and Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY.

Referenced by Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.80 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_no_check (const Congruence_System & *cgs*) [inline, private]

Uses the congruences in `cgs` to refine `*this`.

Parameters

cgs The congruences to be added. Nontrivial proper congruences are ignored.

Warning

If `cgs` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 2288 of file Box.templates.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Congruence_System::space_dimension().

11.13.4.81 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_no_check (const Congruence & *cg*) [inline, private]

Uses the congruence `cg` to refine `*this`.

Parameters

cg The congruence to be added. Nontrivial proper congruences are ignored.

Warning

If `cg` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 2267 of file Box.templates.hh.

References Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Congruence::space_dimension().

11.13.4.82 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_no_check (const Constraint_System & *cs*) [inline, private]

Uses the constraints in `cs` to refine `*this`.

Parameters

cs The constraints to be used for the refinement. To avoid termination problems, each constraint in `cs` will be used for a single refinement step.

Warning

If `cs` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 2257 of file Box.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::space_dimension().

11.13.4.83 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_no_check (const Constraint & *c*) [inline, private]

Uses the constraint `c` to refine `*this`.

Parameters

c The constraint to be used for the refinement.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 2227 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::add_interval_constraint_no_check(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint(), and Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints().

11.13.4.84 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence (const Congruence & *cg*) [inline]

Use the congruence `cg` to refine `*this`.

Parameters

cg The congruence to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cg` are dimension-incompatible.

Definition at line 467 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.85 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Use the congruences in `cgs` to refine `*this`.

Parameters

cgs The congruences to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 482 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.86 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint (const Constraint & *c*) [inline]

Use the constraint `c` to refine `*this`.

Parameters

c The constraint to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible.

Definition at line 438 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.87 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints (const Constraint_System & *cs*) [inline]

Use the constraints in `cs` to refine `*this`.

Parameters

cs The constraints to be used for refinement. To avoid termination problems, each constraint in `cs` will be used for a single refinement step.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Note

The user is warned that the accuracy of this refinement operator depends on the order of evaluation of the constraints in `cs`, which is in general unpredictable. If a fine control on such an order is needed, the user should consider calling the method `refine_with_constraint(const Constraint& c)` inside an appropriate looping construct.

Definition at line 453 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.88 template<typename ITV > Poly_Gen_Relation Parma_Polyhedra_Library::Box< ITV >::relation_with (const Generator & *g*) const [inline]

Returns the relations holding between `*this` and the generator `g`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible.

Definition at line 918 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::Generator::is_ray(),

Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.89 template<typename ITV > Poly_Con_Relation Parma_Polyhedra_Library::Box< ITV >::relation_with (const Congruence & *cg*) const [inline]

Returns the relations holding between `*this` and the congruence `cg`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `cg` are dimension-incompatible.

Definition at line 771 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Congruence::coefficient(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.90 template<typename ITV > Poly_Con_Relation Parma_Polyhedra_Library::Box< ITV >::relation_with (const Constraint & *c*) const [inline]

Returns the relations holding between `*this` and the constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 833 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Box< ITV >::relation_with().

11.13.4.91 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::remove_higher_space_dimensions (dimension_type *new_dimension*) [inline]

Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimension` is greater than the space dimension of `*this`.

Definition at line 2030 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions().

11.13.4.92 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions (const Variables_Set & *vars*) [inline]

Removes all the specified dimensions.

Parameters

vars The set of Variable objects corresponding to the dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 1977 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions().

11.13.4.93 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date () [inline, private]

Invalidates empty flag of `*this`.

Definition at line 64 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::Status::reset_empty_up_to_date(), and Parma_Polyhedra_Library::Box< ITV >::status.

Referenced by Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.94 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::set_empty () [inline]

Causes the box to become empty, i.e., to represent the empty set.

Definition at line 44 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::Status::set_empty(), Parma_Polyhedra_Library::Box< ITV >::Status::set_empty_up_to_date(), and Parma_Polyhedra_Library::Box< ITV >::status.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Box< ITV >::check_empty(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.95 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date () [inline, private]

Asserts the validity of the empty flag of `*this`.

Definition at line 58 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::Status::set_empty_up_to_date(), and Parma_Polyhedra_Library::Box< ITV >::status.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box().

11.13.4.96 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::set_interval (Variable *var*, const ITV & *i*) [inline]

Sets to `i` the interval that bounds `var`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 157 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV

>::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.97 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::set_nonempty () [inline, private]

Marks `*this` as definitely not empty.

Definition at line 51 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::Status::reset_empty(), Parma_Polyhedra_Library::Box< ITV >::Status::set_empty_up_to_date(), and Parma_Polyhedra_Library::Box< ITV >::status.

Referenced by Parma_Polyhedra_Library::Box< ITV >::check_empty(), and Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign().

11.13.4.98 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign (const Box< ITV > & *y*) [inline]

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1863 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_nonempty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.99 template<typename ITV > dimension_type Parma_Polyhedra_Library::Box< ITV >::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 122 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::seq.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::affine_dimension(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Box<

ITV >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::is_disjoint_from(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.4.100 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::strictly_contains (const Box< ITV > & *y*) const [inline]

Returns `true` if and only if `*this` strictly contains `y`.

Exceptions

std::invalid_argument Thrown if `x` and `y` are dimension-incompatible.

Definition at line 225 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::contains().

11.13.4.101 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::swap (Box< ITV > & *y*) [inline]

Swaps `*this` with `y` (`*this` and `y` can be dimension-incompatible).

Definition at line 84 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::status.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions().

11.13.4.102 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_constraint_incompatible (const char * *method*) [inline, static, private]

Definition at line 4027 of file Box.templates.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_assign().

11.13.4.103 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const char * *name_row*, const Linear_Expression & *y*) const [inline, private]

Definition at line 4047 of file Box.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.104 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Generator & *g*) const [inline, private]

Definition at line 4016 of file Box.templates.hh.

References Parma_Polyhedra_Library::Generator::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.105 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Congruence_System & *cgs*) const [inline, private]

Definition at line 4005 of file Box.templates.hh.

References Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.106 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Constraint_System & *cs*) const [inline, private]

Definition at line 3994 of file Box.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.107 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Congruence & *cg*) const [inline, private]

Definition at line 3983 of file Box.templates.hh.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.108 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Constraint & *c*) const [inline, private]

Definition at line 3972 of file Box.templates.hh.

References Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.4.109 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, dimension_type *required_dim*) const [inline, private]

Definition at line 3961 of file Box.templates.hh.

11.13.4.110 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible (const char * *method*, const Box< ITV > & *x*) const [inline, private]

Definition at line 3949 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_constraint(), Parma_Polyhedra_Library::Box< ITV >::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::Box< ITV

>::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_-Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::is_disjoint_from(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_-assign(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign(), and Parma_-Polyhedra_Library::Box< ITV >::upper_bound_assign_if_exact().

11.13.4.111 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_expression_too_complex (const char * *method*, const Linear_Expression & *e*) [inline, static, private]

Definition at line 4036 of file Box.templates.hh.

11.13.4.112 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_generic (const char * *method*, const char * *reason*) [inline, static, private]

Definition at line 4060 of file Box.templates.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_-Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_-Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::fold_-space_dimensions(), and Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage().

11.13.4.113 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::throw_space_dimension_overflow (const char * *method*, const char * *reason*) [inline, static, private]

Definition at line 4069 of file Box.templates.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box().

11.13.4.114 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign (const Box< ITV > & *y*) [inline]

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1939 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.115 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 1481 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), and Parma_Polyhedra_Library::Box< ITV >::seq.

11.13.4.116 template<typename ITV > memory_size_type Parma_Polyhedra_Library::Box< ITV >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 116 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::external_memory_in_bytes().

11.13.4.117 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::unconstrain (const Variables_Set & *vars*) [inline]

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 1447 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.118 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::unconstrain (Variable *var*) [inline]

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 526 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.13.4.119 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign (const Box< ITV > & *y*) [inline]

Assigns to `*this` the smallest box containing the union of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1744 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.120 template<typename ITV > bool Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign_if_exact (const Box< ITV > & *y*) [inline]

If the upper bound of `*this` and `y` is exact, it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1196 of file Box.templates.hh.

References Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.13.4.121 template<typename T > void Parma_Polyhedra_Library::Box< T >::widening_assign (const Box< ITV > & *y*, unsigned * *tp* = 0) [inline]

Same as CC76_widening_assign(y, tp).

Definition at line 364 of file Box.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign().

11.13.4.122 template<typename ITV > void Parma_Polyhedra_Library::Box< ITV >::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true) [inline]

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system. When non-null, the pointed-to constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in `*pcs`.

complexity_threshold A precision parameter which is ignored for the Box domain.

wrap_individually A precision parameter which is ignored for the Box domain.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars` or with `*pcs`.

Definition at line 1491 of file Box.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Coefficient_one(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::mul_2exp_assign(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::OVERFLOW_IMPOSSIBLE, Parma_Polyhedra_Library::OVERFLOW_UNDEFINED, Parma_Polyhedra_Library::OVERFLOW_WRAPS, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Box< ITV >::refine_interval_no_check(), Parma_Polyhedra_Library::Box< ITV

>::refine_with_constraints(), Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::seq, Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::SIGNED_2_COMPLEMENT, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Constraint::type(), Parma_Polyhedra_Library::UNIVERSE, and Parma_Polyhedra_Library::UNSIGNED.

11.13.5 Friends And Related Function Documentation

11.13.5.1 template<typename Temp , typename To , typename ITV > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 591 of file Box.inlines.hh.

11.13.5.2 template<typename To , typename ITV > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 605 of file Box.inlines.hh.

11.13.5.3 template<typename ITV> bool extract_interval_constraint (const Constraint & *c*, dimension_type *c_space_dim*, dimension_type & *c_num_vars*, dimension_type & *c_only_var*) [related]

Decodes the constraint c as an interval constraint.

Returns

true if the constraint c is an interval constraint; false otherwise.

Parameters

c The constraint to be decoded.

c_space_dim The space dimension of the constraint c (it is *assumed* to match the actual space dimension of c).

c_num_vars If true is returned, then it will be set to the number of variables having a non-zero coefficient. The only legal values will therefore be 0 and 1.

c_only_var If true is returned and if c_num_vars is not set to 0, then it will be set to the index of the only variable having a non-zero coefficient in c.

Definition at line 33 of file Box.cc.

References Parma_Polyhedra_Library::Constraint::coefficient(), and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.5.4 template<typename Temp , typename To , typename ITV > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between x and y.

If the L_∞ distance between x and y is defined, stores an approximation of it into r and returns true; returns false otherwise.

The direction of the approximation is specified by dir.

All computations are performed using the temporary variables tmp0, tmp1 and tmp2.

Definition at line 632 of file Box.inlines.hh.

11.13.5.5 template<typename To , typename ITV > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*) [related]

Computes the L_∞ distance between x and y.

If the L_∞ distance between x and y is defined, stores an approximation of it into r and returns true; returns false otherwise.

The direction of the approximation is specified by dir.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 646 of file Box.inlines.hh.

11.13.5.6 template<typename Specialization , typename Temp , typename To , typename ITV > bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Helper function for computing distances.

Definition at line 4083 of file Box.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::finalize(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::maybe_assign(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Box< ITV >::seq, and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.13.5.7 template<typename ITV > bool operator!= (const Box< ITV > & *x*, const Box< ITV > & *y*) [related]

Returns `true` if and only if `x` and `y` aren't the same box.

Note that `x` and `y` may be dimension-incompatible boxes: in this case, the value `true` is returned.

Definition at line 254 of file Box.inlines.hh.

11.13.5.8 template<typename ITV > std::ostream & operator<< (std::ostream & *s*, const Box< ITV > & *box*) [related]

Output operator.

Definition at line 3888 of file Box.templates.hh.

11.13.5.9 template<typename ITV> bool operator== (const Box< ITV > & *x*, const Box< ITV > & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same box.

Note that `x` and `y` may be dimension-incompatible boxes: in this case, the value `false` is returned.

Definition at line 533 of file Box.templates.hh.

11.13.5.10 template<typename ITV> friend class Parma_Polyhedra_Library::Box [friend]

Definition at line 1677 of file Box.defs.hh.

11.13.5.11 template<typename ITV> std::ostream& Parma_Polyhedra_Library::IO_-Operators::operator<< (std::ostream & *s*, const Box< ITV > & *box*) [friend]

11.13.5.12 template<typename ITV> template<typename Specialization , typename Temp , typename To , typename I > bool Parma_Polyhedra_Library::l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< I > & *x*, const Box< I > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.13.5.13 template<typename Temp , typename To , typename ITV > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 550 of file Box.inlines.hh.

11.13.5.14 template<typename To , typename ITV > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Box< ITV > & *x*, const Box< ITV > & *y*, Rounding_Dir *dir*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_-Policy>.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 564 of file Box.inlines.hh.

11.13.6 Member Data Documentation

11.13.6.1 template<typename ITV> Sequence Parma_Polyhedra_Library::Box< ITV >::seq [private]

A sequence of intervals, one for each dimension of the vector space.

Definition at line 1702 of file Box.defs.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Box< ITV >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Box< ITV >::affine_dimension(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::ascii_dump(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Box< ITV >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Box< ITV >::CC76_widening_assign(), Parma_Polyhedra_Library::Box< ITV >::check_empty(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::Box< ITV >::contains(), Parma_Polyhedra_Library::Box< ITV >::contains_integer_point(), Parma_Polyhedra_Library::Box< ITV >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::external_memory_in_bytes(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(), Parma_Polyhedra_Library::Box< ITV >::intersection_assign(), Parma_Polyhedra_Library::Box< ITV >::is_bounded(), Parma_Polyhedra_Library::Box< ITV >::is_discrete(), Parma_Polyhedra_Library::Box< ITV >::is_disjoint_from(), Parma_Polyhedra_Library::Box< ITV >::is_topologically_closed(), Parma_Polyhedra_Library::Box< ITV >::is_universe(), Parma_Polyhedra_Library::Box< ITV >::l_m_distance_assign(), Parma_Polyhedra_Library::Box< ITV >::map_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::operator=(), Parma_Polyhedra_Library::operator==(), Parma_Polyhedra_Library::Box< ITV >::operator[ ](), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Box< ITV >::simplify_using_context_assign(), Parma_Polyhedra_Library::Box< ITV >::space_dimension(), Parma_Polyhedra_Library::Box< ITV >::swap(), Parma_Polyhedra_Library::Box< ITV >::time_elapse_assign(), Parma_Polyhedra_Library::Box< ITV >::topological_closure_assign(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign(), Parma_Polyhedra_Library::Box< ITV >::upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.13.6.2 template<typename ITV> Status Parma_Polyhedra_Library::Box< ITV >::status [private]

The status flags to keep track of the internal state.

Definition at line 1709 of file Box.defs.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::ascii_dump(), Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), Parma_Polyhedra_Library::Box< ITV >::OK(), Parma_Polyhedra_Library::Box< ITV >::operator=(), Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), Parma_Polyhedra_Library::Box< ITV >::set_nonempty(), and Parma_Polyhedra_Library::Box< ITV >::swap().

The documentation for this class was generated from the following files:

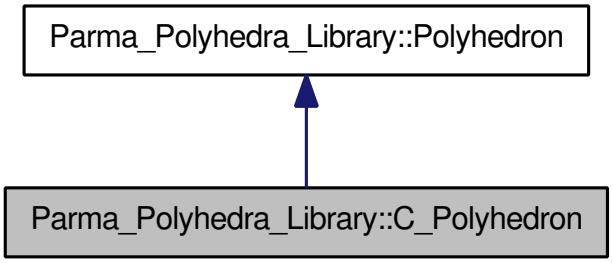
- Box.defs.hh
- Box.inlines.hh
- Box.templates.hh
- Box.cc

11.14 Parma_Polyhedra_Library::C_Polyhedron Class Reference

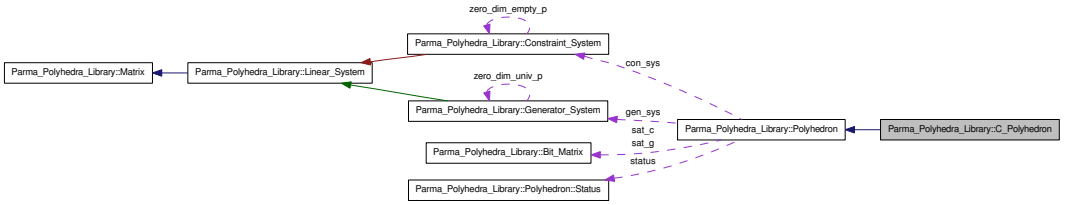
A closed convex polyhedron.

`#include <C_Polyhedron.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::C_Polyhedron:



Collaboration diagram for Parma_Polyhedra_Library::C_Polyhedron:



Public Member Functions

- C_Polyhedron (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

Builds either the universe or the empty C polyhedron.

- C_Polyhedron (const Constraint_System &cs)

  Builds a C polyhedron from a system of constraints.

- C_Polyhedron (Constraint_System &cs, Recycle_Input dummy)

  Builds a C polyhedron recycling a system of constraints.

- C_Polyhedron (const Generator_System &gs)

  Builds a C polyhedron from a system of generators.

- C_Polyhedron (Generator_System &gs, Recycle_Input dummy)

  Builds a C polyhedron recycling a system of generators.

- C_Polyhedron (const Congruence_System &cgs)

  Builds a C polyhedron from a system of congruences.

- C_Polyhedron (Congruence_System &cgs, Recycle_Input dummy)

  Builds a C polyhedron recycling a system of congruences.

- C_Polyhedron (const NNC_Polyhedron &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a C polyhedron representing the topological closure of the NNC polyhedron `y`.

- template<typename Interval >
  C_Polyhedron (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a C polyhedron out of a box.

- template<typename U >
  C_Polyhedron (const BD_Shape< U > &bd, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a C polyhedron out of a BD shape.

- template<typename U >
  C_Polyhedron (const Octagonal_Shape< U > &os, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a C polyhedron out of an octagonal shape.

- C_Polyhedron (const Grid &grid, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a C polyhedron out of a grid.

- C_Polyhedron (const C_Polyhedron &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- C_Polyhedron & operator= (const C_Polyhedron &y)

  The assignment operator. (`*this` *and* `y` *can be dimension-incompatible.).*

- C_Polyhedron & operator= (const NNC_Polyhedron &y)

  Assigns to `*this` *the topological closure of the NNC polyhedron* `y`.

- ~C_Polyhedron ()

  Destructor.

- bool poly_hull_assign_if_exact (const C_Polyhedron &y)

  If the poly-hull of `*this` *and* `y` *is exact it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- bool upper_bound_assign_if_exact (const C_Polyhedron &y)

  Same as poly_hull_assign_if_exact(y).

11.14.1 Detailed Description

A closed convex polyhedron. An object of the class C_Polyhedron represents a *topologically closed* convex polyhedron in the vector space $\mathbb{R}^n$.

When building a closed polyhedron starting from a system of constraints, an exception is thrown if the system contains a *strict inequality* constraint. Similarly, an exception is thrown when building a closed polyhedron starting from a system of generators containing a *closure point*.

Note

Such an exception will be obtained even if the system of constraints (resp., generators) actually defines a topologically closed subset of the vector space, i.e., even if all the strict inequalities (resp., closure points) in the system happen to be redundant with respect to the system obtained by removing all the strict inequality constraints (resp., all the closure points). In contrast, when building a closed polyhedron starting from an object of the class NNC_Polyhedron, the precise topological closure test will be performed.

Definition at line 59 of file C_Polyhedron.defs.hh.

11.14.2 Constructor & Destructor Documentation

11.14.2.1 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds either the universe or the empty C polyhedron.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the C polyhedron;

kind Specifies whether a universe or an empty C polyhedron should be built.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Both parameters are optional: by default, a 0-dimension space universe C polyhedron is built.

Definition at line 37 of file C_Polyhedron.inlines.hh.

11.14.2.2 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Constraint_System & *cs*) [inline, explicit]

Builds a C polyhedron from a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the polyhedron.

Exceptions

std::invalid_argument Thrown if the system of constraints contains strict inequalities.

Definition at line 51 of file C_Polyhedron.inlines.hh.

11.14.2.3 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (Constraint_System & *cs*, Recycle_Input *dummy*) [inline]

Builds a C polyhedron recycling a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the system of constraints contains strict inequalities.

Definition at line 63 of file C_Polyhedron.inlines.hh.

11.14.2.4 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Generator_System & *gs*) [inline, explicit]

Builds a C polyhedron from a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

gs The system of generators defining the polyhedron.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points, or if it contains closure points.

Definition at line 76 of file C_Polyhedron.inlines.hh.

11.14.2.5 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (Generator_System & *gs*, Recycle_Input *dummy*) [inline]

Builds a C polyhedron recycling a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

gs The system of generators defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points, or if it contains closure points.

Definition at line 88 of file C_Polyhedron.inlines.hh.

11.14.2.6 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Congruence_System & *cgs*) [explicit]

Builds a C polyhedron from a system of congruences.

The polyhedron inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the polyhedron.

Definition at line 44 of file C_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_congruences().

11.14.2.7 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (Congruence_System & *cgs*, Recycle_Input *dummy*)

Builds a C polyhedron recycling a system of congruences.

The polyhedron inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Definition at line 57 of file C_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_congruences().

11.14.2.8 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const NNC_Polyhedron & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [explicit]

Builds a C polyhedron representing the topological closure of the NNC polyhedron `y`.

Parameters

y The NNC polyhedron to be used;

complexity This argument is ignored.

Definition at line 33 of file C_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.14.2.9 template<typename Interval > Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a C polyhedron out of a box.

The polyhedron inherits the space dimension of the box and is the most precise that includes the box. The algorithm used has polynomial complexity.

Parameters

box The box representing the polyhedron to be approximated;

complexity This argument is ignored.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 102 of file C_Polyhedron.inlines.hh.

11.14.2.10 template<typename U > Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const BD_Shape< U > & *bd*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a C polyhedron out of a BD shape.

The polyhedron inherits the space dimension of the BDS and is the most precise that includes the BDS.

Parameters

bd The BDS used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 115 of file C_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::constraints().

11.14.2.11 template<typename U > Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Octagonal_Shape< U > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a C polyhedron out of an octagonal shape.

The polyhedron inherits the space dimension of the octagonal shape and is the most precise that includes the octagonal shape.

Parameters

os The octagonal shape used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 130 of file C_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints().

11.14.2.12 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const Grid & *grid*, Complexity_Class *complexity* = ANY_COMPLEXITY) [explicit]

Builds a C polyhedron out of a grid.

The polyhedron inherits the space dimension of the grid and is the most precise that includes the grid.

Parameters

grid The grid used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 71 of file C_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::Grid::constraints().

11.14.2.13 Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron (const C_Polyhedron & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 144 of file C_Polyhedron.inlines.hh.

11.14.2.14 Parma_Polyhedra_Library::C_Polyhedron::~C_Polyhedron () [inline]

Destructor.

Definition at line 33 of file C_Polyhedron.inlines.hh.

11.14.3 Member Function Documentation

11.14.3.1 C_Polyhedron & Parma_Polyhedra_Library::C_Polyhedron::operator= (const NNC_Polyhedron & *y*) [inline]

Assigns to `*this` the topological closure of the NNC polyhedron `y`.

Definition at line 155 of file C_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::swap().

11.14.3.2 C_Polyhedron & Parma_Polyhedra_Library::C_Polyhedron::operator= (const C_Polyhedron & *y*) [inline]

The assignment operator. (`*this` and `y` can be dimension-incompatible.).

Definition at line 149 of file C_Polyhedron.inlines.hh.

11.14.3.3 bool Parma_Polyhedra_Library::C_Polyhedron::poly_hull_assign_if_exact (const C_Polyhedron & *y*)

If the poly-hull of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 85 of file C_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::space_dimension(), and Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible().

Referenced by upper_bound_assign_if_exact().

11.14.3.4 bool Parma_Polyhedra_Library::C_Polyhedron::upper_bound_assign_if_exact (const C_Polyhedron & *y*) [inline]

Same as poly_hull_assign_if_exact(y).

Definition at line 162 of file C_Polyhedron.inlines.hh.

References poly_hull_assign_if_exact().

The documentation for this class was generated from the following files:

- C_Polyhedron.defs.hh
- C_Polyhedron.inlines.hh
- C_Polyhedron.cc

11.15 Parma_Polyhedra_Library::Check_Overflow_Policy< T > Struct Template Reference

A policy checking for overflows.

`#include <Checked_Number.defs.hh>`

Public Member Functions

- const_bool_nodef (check_overflow, true)
- const_bool_nodef (check_inf_add_inf, false)
- const_bool_nodef (check_inf_sub_inf, false)
- const_bool_nodef (check_inf_mul_zero, false)
- const_bool_nodef (check_div_zero, false)
- const_bool_nodef (check_inf_div_inf, false)
- const_bool_nodef (check_inf_mod, false)
- const_bool_nodef (check_sqrt_neg, false)
- const_bool_nodef (has_nan, std::numeric_limits< T >::has_quiet_NaN)
- const_bool_nodef (has_infinity, std::numeric_limits< T >::has_infinity)
- const_bool_nodef (convertible, true)
- const_bool_nodef (fpu_check_inexact, true)
- const_bool_nodef (fpu_check_nan_result, true)

11.15.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Check_Overflow_Policy< T >

A policy checking for overflows.

Definition at line 65 of file Checked_Number.defs.hh.

11.15.2 Member Function Documentation

11.15.2.1 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (fpu_check_nan_result, true)

11.15.2.2 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (fpu_check_inexact, true)

11.15.2.3 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (convertible, true)

11.15.2.4 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (has_infinity, std::numeric_limits< T >::has_infinity)

11.15.2.5 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (has_nan, std::numeric_limits< T >::has_quiet_NaN)

11.15.2.6 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_sqrt_neg, false)

11.15.2.7 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_inf_mod, false)

11.15.2.8 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_inf_div_inf, false)

11.15.2.9 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_div_zero, false)

11.15.2.10 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_inf_mul_zero, false)

11.15.2.11 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_inf_sub_inf, false)

11.15.2.12 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_inf_add_inf, false)

11.15.2.13 template<typename T > Parma_Polyhedra_Library::Check_Overflow_Policy< T >::const_bool_nodef (check_overflow, true)

The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

11.16 Parma_Polyhedra_Library::Checked_Number< T, Policy > Class Template Reference

A wrapper for numeric types implementing a given policy.

`#include <Checked_Number.defs.hh>`

Public Member Functions

- bool OK () const
  Checks if all the invariants are satisfied.
- Result classify (bool nan=true, bool inf=true, bool sign=true) const
  Classifies ∗this.

Constructors

- Checked_Number ()
  Default constructor.
- Checked_Number (const Checked_Number &y)
  Copy constructor.
- template<typename From , typename From_Policy >
  Checked_Number (const Checked_Number< From, From_Policy > &y, Rounding_Dir dir)
  Direct initialization from a Checked_Number and rounding mode.
- Checked_Number (char y, Rounding_Dir dir)
  Direct initialization from a plain char and rounding mode.
- Checked_Number (signed char y, Rounding_Dir dir)
  Direct initialization from a signed char and rounding mode.
- Checked_Number (signed short y, Rounding_Dir dir)
  Direct initialization from a signed short and rounding mode.
- Checked_Number (signed int y, Rounding_Dir dir)

Direct initialization from a signed int and rounding mode.

- Checked_Number (signed long y, Rounding_Dir dir)
  Direct initialization from a signed long and rounding mode.
- Checked_Number (signed long long y, Rounding_Dir dir)
  Direct initialization from a signed long long and rounding mode.
- Checked_Number (unsigned char y, Rounding_Dir dir)
  Direct initialization from an unsigned char and rounding mode.
- Checked_Number (unsigned short y, Rounding_Dir dir)
  Direct initialization from an unsigned short and rounding mode.
- Checked_Number (unsigned int y, Rounding_Dir dir)
  Direct initialization from an unsigned int and rounding mode.
- Checked_Number (unsigned long y, Rounding_Dir dir)
  Direct initialization from an unsigned long and rounding mode.
- Checked_Number (unsigned long long y, Rounding_Dir dir)
  Direct initialization from an unsigned long long and rounding mode.
- Checked_Number (const mpq_class &y, Rounding_Dir dir)
  Direct initialization from a rational and rounding mode.
- Checked_Number (const mpz_class &y, Rounding_Dir dir)
  Direct initialization from an unbounded integer and rounding mode.
- Checked_Number (const char *y, Rounding_Dir dir)
  Direct initialization from a C string and rounding mode.
- template<typename From >
  Checked_Number (const From &, Rounding_Dir dir, typename Enable_If< Is_Special< From >::value, bool >::type ignored=false)
  Direct initialization from special and rounding mode.
- template<typename From , typename From_Policy >
  Checked_Number (const Checked_Number< From, From_Policy > &y)
  Direct initialization from a Checked_Number, default rounding mode.
- Checked_Number (char y)
  Direct initialization from a plain char, default rounding mode.
- Checked_Number (signed char y)
  Direct initialization from a signed char, default rounding mode.
- Checked_Number (signed short y)
  Direct initialization from a signed short, default rounding mode.
- Checked_Number (signed int y)
  Direct initialization from a signed int, default rounding mode.
- Checked_Number (signed long y)
  Direct initialization from a signed long, default rounding mode.

- Checked_Number (signed long long y)
  Direct initialization from a signed long long, default rounding mode.
- Checked_Number (unsigned char y)
  Direct initialization from an unsigned char, default rounding mode.
- Checked_Number (unsigned short y)
  Direct initialization from an unsigned short, default rounding mode.
- Checked_Number (unsigned int y)
  Direct initialization from an unsigned int, default rounding mode.
- Checked_Number (unsigned long y)
  Direct initialization from an unsigned long, default rounding mode.
- Checked_Number (unsigned long long y)
  Direct initialization from an unsigned long long, default rounding mode.
- Checked_Number (float y)
  Direct initialization from a float, default rounding mode.
- Checked_Number (double y)
  Direct initialization from a double, default rounding mode.
- Checked_Number (long double y)
  Direct initialization from a long double, default rounding mode.
- Checked_Number (const mpq_class &y)
  Direct initialization from a rational, default rounding mode.
- Checked_Number (const mpz_class &y)
  Direct initialization from an unbounded integer, default rounding mode.
- Checked_Number (const char *y)
  Direct initialization from a C string, default rounding mode.
- template<typename From >
  Checked_Number (const From &, typename Enable_If< Is_Special< From >::value, bool >::type ignored=false)
  Direct initialization from special, default rounding mode.

Accessors and Conversions

- operator T () const
  Conversion operator: returns a copy of the underlying numeric value.
- T & raw_value ()
  Returns a reference to the underlying numeric value.
- const T & raw_value () const
  Returns a const reference to the underlying numeric value.

Assignment Operators

- Checked_Number & operator= (const Checked_Number &y)
  Assignment operator.

- template<typename From >
  Checked_Number & operator= (const From &y)
  Assignment operator.

- template<typename From_Policy >
  Checked_Number & operator+= (const Checked_Number< T, From_Policy > &y)
  Add and assign operator.

- Checked_Number & operator+= (const T &y)
  Add and assign operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, Checked_Number< T, Policy > & >::type
  operator+= (const From &y)
  Add and assign operator.

- template<typename From_Policy >
  Checked_Number & operator-= (const Checked_Number< T, From_Policy > &y)
  Subtract and assign operator.

- Checked_Number & operator-= (const T &y)
  Subtract and assign operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, Checked_Number< T, Policy > & >::type
  operator-= (const From &y)
  Subtract and assign operator.

- template<typename From_Policy >
  Checked_Number & operator*= (const Checked_Number< T, From_Policy > &y)
  Multiply and assign operator.

- Checked_Number & operator*= (const T &y)
  Multiply and assign operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, Checked_Number< T, Policy > & >::type
  operator*= (const From &y)
  Multiply and assign operator.

- template<typename From_Policy >
  Checked_Number & operator/= (const Checked_Number< T, From_Policy > &y)
  Divide and assign operator.

- Checked_Number & operator/= (const T &y)
  Divide and assign operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, Checked_Number< T, Policy > & >::type
  operator/= (const From &y)
  Divide and assign operator.

- template<typename From_Policy >
  Checked_Number & operator%= (const Checked_Number< T, From_Policy > &y)
  Compute remainder and assign operator.

- Checked_Number & operator%= (const T &y)
  Compute remainder and assign operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, Checked_Number< T, Policy > & >::type operator%= (const From &y)
  Compute remainder and assign operator.

Increment and Decrement Operators

- Checked_Number & operator++ ()
  Pre-increment operator.

- Checked_Number operator++ (int)
  Post-increment operator.

- Checked_Number & operator-- ()
  Pre-decrement operator.

- Checked_Number operator-- (int)
  Post-decrement operator.

Private Attributes

- T v
  The underlying numeric value.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_not_a_number (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_minus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_plus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, int >::type is_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_integer (const T &x)
- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type construct (To &to, const From &x, Rounding_Dir dir)

- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type assign_r (To &to, const From &x, Rounding_Dir dir)
- template<typename To >
  Enable_If< Is_Native_Or_Checked< To >::value, Result >::type assign_r (To &to, const char *x, Rounding_Dir dir)
- template<typename To , typename To_Policy >
  Enable_If< Is_Native_Or_Checked< To >::value, Result >::type assign_r (To &to, char *x, Rounding_Dir dir)
- template<typename T , typename Policy >
  void swap (Checked_Number< T, Policy > &x, Checked_Number< T, Policy > &y)

  Swaps x with y.

- template<typename T , typename Policy >
  const T & raw_value (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  T & raw_value (Checked_Number< T, Policy > &x)

Memory Size Inspection Functions

- template<typename T , typename Policy >
  size_t total_memory_in_bytes (const Checked_Number< T, Policy > &x)

  Returns the total size in bytes of the memory occupied by x.

- template<typename T , typename Policy >
  memory_size_type external_memory_in_bytes (const Checked_Number< T, Policy > &x)

  Returns the size in bytes of the memory managed by x.

Arithmetic Operators

- template<typename T , typename Policy >
  Checked_Number< T, Policy > operator+ (const Checked_Number< T, Policy > &x)

  Unary plus operator.

- template<typename T , typename Policy >
  Checked_Number< T, Policy > operator- (const Checked_Number< T, Policy > &x)

  Unary minus operator.

- template<typename T , typename Policy >
  void floor_assign (Checked_Number< T, Policy > &x)

  Assigns to x largest integral value not greater than x.

- template<typename T , typename Policy >
  void floor_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)

  Assigns to x largest integral value not greater than y.

- template<typename T , typename Policy >
  void ceil_assign (Checked_Number< T, Policy > &x)

  Assigns to x smallest integral value not less than x.

- template<typename T , typename Policy >
  void ceil_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)

Assigns to `x` *smallest integral value not less than* `y`.

- template<typename T , typename Policy >
  void trunc_assign (Checked_Number< T, Policy > &x)
  Round `x` *to the nearest integer not larger in absolute value.*

- template<typename T , typename Policy >
  void trunc_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)
  Assigns to `x` *the value of* `y` *rounded to the nearest integer not larger in absolute value.*

- template<typename T , typename Policy >
  void neg_assign (Checked_Number< T, Policy > &x)
  Assigns to `x` *its negation.*

- template<typename T , typename Policy >
  void neg_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)
  Assigns to `x` *the negation of* `y`.

- template<typename T , typename Policy >
  void abs_assign (Checked_Number< T, Policy > &x)
  Assigns to `x` *its absolute value.*

- template<typename T , typename Policy >
  void abs_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)
  Assigns to `x` *the absolute value of* `y`.

- template<typename T , typename Policy >
  void add_mul_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
  Assigns to `x` *the value* `x + y * z`.

- template<typename T , typename Policy >
  void sub_mul_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
  Assigns to `x` *the value* `x - y * z`.

- template<typename T , typename Policy >
  void gcd_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
  Assigns to `x` *the greatest common divisor of* `y` *and* `z`.

- template<typename T , typename Policy >
  void gcdext_assign (Checked_Number< T, Policy > &x, Checked_Number< T, Policy > &s, Checked_Number< T, Policy > &t, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
  Assigns to `x` *the greatest common divisor of* `y` *and* `z`*, setting* `s` *and* `t` *such that* $s*y + t*z = x = gcd(y, z)$.

- template<typename T , typename Policy >
  void lcm_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
  Assigns to `x` *the least common multiple of* `y` *and* `z`.

- template<typename T , typename Policy >
  void mul_2exp_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, unsigned int exp)

  Assigns to x the value $y \cdot 2^{\mathtt{exp}}$.

- template<typename T , typename Policy >
  void div_2exp_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, unsigned int exp)

  Assigns to x the value $y/2^{\mathtt{exp}}$.

- template<typename T , typename Policy >
  void exact_div_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)

  If z divides y, assigns to x the quotient of the integer division of y and z.

- template<typename T , typename Policy >
  void sqrt_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y)

  Assigns to x the integer square root of y.

Relational Operators and Comparison Functions

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator== (const T1 &x, const T2 &y)

  Equality operator.

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator!= (const T1 &x, const T2 &y)

  Disequality operator.

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator>= (const T1 &x, const T2 &y)

  Greater than or equal to operator.

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator> (const T1 &x, const T2 &y)

  Greater than operator.

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator<= (const T1 &x, const T2 &y)

  Less than or equal to operator.

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator< (const T1 &x, const T2 &y)
  Less than operator.

- template<typename From >
  Enable_If< Is_Native_Or_Checked< From >::value, int >::type sgn (const From &x)
  Returns -1, 0 or 1 depending on whether the value of x is negative, zero or positive, respectively.

- template<typename From1 , typename From2 >
  Enable_If< Is_Native_Or_Checked< From1 >::value &&Is_Native_Or_Checked< From2 >::value, int >::type cmp (const From1 &x, const From2 &y)
  Returns a negative, zero or positive value depending on whether x is lower than, equal to or greater than y, respectively.

Input-Output Operators

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, Result >::type output (std::ostream &os, const T &x, const Numeric_Format &fmt, Rounding_Dir dir)
- template<typename T , typename Policy >
  std::ostream & operator<< (std::ostream &os, const Checked_Number< T, Policy > &x)
  Output operator.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, Result >::type input (T &x, std::istream &is, Rounding_Dir dir)
  Input function.

- template<typename T , typename Policy >
  std::istream & operator>> (std::istream &is, Checked_Number< T, Policy > &x)
  Input operator.

11.16.1 Detailed Description

template<typename T, typename Policy> class Parma_Polyhedra_Library::Checked_Number< T, Policy >

A wrapper for numeric types implementing a given policy. The wrapper and related functions implement an interface which is common to all kinds of coefficient types, therefore allowing for a uniform coding style. This class also implements the policy encoded by the second template parameter. The default policy is to perform the detection of overflow errors.

Definition at line 159 of file Checked_Number.defs.hh.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 template<typename T , typename Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number () [inline]

Default constructor.

Definition at line 72 of file Checked_Number.inlines.hh.

11.16.2.2 template<typename T , typename Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const Checked_Number< T, Policy > & *y*) [inline]

Copy constructor.

Definition at line 78 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.2.3 template<typename T , typename Policy > template<typename From , typename From_Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const Checked_Number< From, From_Policy > & *y*, Rounding_Dir *dir*) [inline]

Direct initialization from a Checked_Number and rounding mode.

Definition at line 87 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::rounding_dir().

11.16.2.4 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (char *y*, Rounding_Dir *dir*)

Direct initialization from a plain char and rounding mode.

11.16.2.5 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed char *y*, Rounding_Dir *dir*)

Direct initialization from a signed char and rounding mode.

11.16.2.6 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed short *y*, Rounding_Dir *dir*)

Direct initialization from a signed short and rounding mode.

11.16.2.7 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed int *y*, Rounding_Dir *dir*)

Direct initialization from a signed int and rounding mode.

11.16.2.8 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed long *y*, Rounding_Dir *dir*)

Direct initialization from a signed long and rounding mode.

11.16.2.9 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed long long *y*, Rounding_Dir *dir*)

Direct initialization from a signed long long and rounding mode.

11.16.2.10 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned char *y*, Rounding_Dir *dir*)

Direct initialization from an unsigned char and rounding mode.

11.16.2.11 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned short *y*, Rounding_Dir *dir*)

Direct initialization from an unsigned short and rounding mode.

11.16.2.12 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned int *y*, Rounding_Dir *dir*)

Direct initialization from an unsigned int and rounding mode.

11.16.2.13 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned long *y*, Rounding_Dir *dir*)

Direct initialization from an unsigned long and rounding mode.

11.16.2.14 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned long long *y*, Rounding_Dir *dir*)

Direct initialization from an unsigned long long and rounding mode.

11.16.2.15 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const mpq_class & *y*, Rounding_Dir *dir*)

Direct initialization from a rational and rounding mode.

11.16.2.16 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const mpz_class & *y*, Rounding_Dir *dir*)

Direct initialization from an unbounded integer and rounding mode.

11.16.2.17 template<typename T , typename Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const char * *y*, Rounding_Dir *dir*) [inline]

Direct initialization from a C string and rounding mode.

Definition at line 163 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::rounding_dir(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.2.18 template<typename T , typename Policy > template<typename From > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const From &, Rounding_Dir *dir*, typename Enable_If< Is_Special< From >::value, bool >::type *ignored* = false) [inline]

Direct initialization from special and rounding mode.

Definition at line 185 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::rounding_dir(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.2.19 template<typename T , typename Policy > template<typename From , typename From_Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const Checked_Number< From, From_Policy > & *y*) [inline, explicit]

Direct initialization from a Checked_Number, default rounding mode.

Definition at line 102 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::rounding_dir().

11.16.2.20 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (char *y*)

Direct initialization from a plain char, default rounding mode.

11.16.2.21 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed char *y*)

Direct initialization from a signed char, default rounding mode.

11.16.2.22 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed short *y*)

Direct initialization from a signed short, default rounding mode.

11.16.2.23 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed int *y*)

Direct initialization from a signed int, default rounding mode.

11.16.2.24 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed long *y*)

Direct initialization from a signed long, default rounding mode.

11.16.2.25 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (signed long long *y*)

Direct initialization from a signed long long, default rounding mode.

11.16.2.26 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned char *y*)

Direct initialization from an unsigned char, default rounding mode.

11.16.2.27 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned short *y*)

Direct initialization from an unsigned short, default rounding mode.

11.16.2.28 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned int *y*)

Direct initialization from an unsigned int, default rounding mode.

11.16.2.29 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned long *y*)

Direct initialization from an unsigned long, default rounding mode.

11.16.2.30 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (unsigned long long *y*)

Direct initialization from an unsigned long long, default rounding mode.

11.16.2.31 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (float *y*)

Direct initialization from a float, default rounding mode.

11.16.2.32 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (double *y*)

Direct initialization from a double, default rounding mode.

11.16.2.33 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (long double *y*)

Direct initialization from a long double, default rounding mode.

11.16.2.34 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const mpq_class & *y*)

Direct initialization from a rational, default rounding mode.

11.16.2.35 template<typename T, typename Policy> Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const mpz_class & *y*)

Direct initialization from an unbounded integer, default rounding mode.

11.16.2.36 template<typename T , typename Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const char * *y*) [inline]

Direct initialization from a C string, default rounding mode.

Definition at line 173 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::rounding_dir(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.2.37 template<typename T , typename Policy > template<typename From > Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number (const From &, typename Enable_If< Is_Special< From >::value, bool >::type *ignored* = false) [inline]

Direct initialization from special, default rounding mode.

Definition at line 195 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::rounding_dir(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.3 Member Function Documentation

11.16.3.1 template<typename T , typename Policy > Result Parma_Polyhedra_Library::Checked_Number< T, Policy >::classify (bool *nan* = true, bool *inf* = true, bool *sign* = true) const [inline]

Classifies ∗this.

Returns the appropriate Result characterizing:

- whether `*this` is NaN, if `nan` is `true`;
- whether `*this` is a (positive or negative) infinity, if `inf` is `true`;
- the sign of `*this`, if `sign` is `true`.

Definition at line 298 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.3.2 template<typename T , typename Policy > bool Parma_Polyhedra_Library::Checked_Number< T, Policy >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 292 of file Checked_Number.inlines.hh.

11.16.3.3 template<typename T , typename Policy > Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator T () const [inline]

Conversion operator: returns a copy of the underlying numeric value.

Definition at line 259 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.3.4 template<typename T, typename Policy> template<typename From > Enable_If<Is_Native_Or_Checked<From>::value, Checked_Number<T, Policy>& >::type Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator%= (const From & *y*) [inline]

Compute remainder and assign operator.

11.16.3.5 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator%= (const T & *y*)

Compute remainder and assign operator.

11.16.3.6 template<typename T, typename Policy> template<typename From_Policy > Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator%= (const Checked_Number< T, From_Policy > & *y*) [inline]

Compute remainder and assign operator.

11.16.3.7 template<typename T, typename Policy> template<typename From > Enable_If<Is_Native_Or_Checked<From>::value, Checked_Number<T, Policy>&>::type Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator*= (const From & *y*) [inline]

Multiply and assign operator.

11.16.3.8 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator*= (const T & *y*)

Multiply and assign operator.

11.16.3.9 template<typename T, typename Policy> template<typename From_Policy > Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator*= (const Checked_Number< T, From_Policy > & *y*) [inline]

Multiply and assign operator.

11.16.3.10 template<typename T, typename Policy> Checked_Number Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator++ (int)

Post-increment operator.

11.16.3.11 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator++ ()

Pre-increment operator.

11.16.3.12 template<typename T, typename Policy> template<typename From > Enable_If<Is_Native_Or_Checked<From>::value, Checked_Number<T, Policy>&>::type Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator+= (const From & *y*) [inline]

Add and assign operator.

11.16.3.13 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator+= (const T & *y*)

Add and assign operator.

11.16.3.14 template<typename T, typename Policy> template<typename From_Policy > Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator+= (const Checked_Number< T, From_Policy > & *y*) [inline]

Add and assign operator.

11.16.3.15 template<typename T, typename Policy> Checked_Number Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator-- (int)

Post-decrement operator.

11.16.3.16 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator-- ()

Pre-decrement operator.

11.16.3.17 template<typename T, typename Policy> template<typename From > Enable_If<Is_Native_Or_Checked<From>::value, Checked_Number<T, Policy>&>::type Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator-= (const From & *y*) [inline]

Subtract and assign operator.

11.16.3.18 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator-= (const T & *y*)

Subtract and assign operator.

11.16.3.19 template<typename T, typename Policy> template<typename From_Policy > Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator-= (const Checked_Number< T, From_Policy > & *y*) [inline]

Subtract and assign operator.

11.16.3.20 template<typename T, typename Policy> template<typename From > Enable_If<Is_Native_Or_Checked<From>::value, Checked_Number<T, Policy>&>::type Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator/= (const From & *y*) [inline]

Divide and assign operator.

11.16.3.21 template<typename T, typename Policy> Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator/= (const T & *y*)

Divide and assign operator.

11.16.3.22 template<typename T, typename Policy> template<typename From_Policy > Checked_Number& Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator/= (const Checked_Number< T, From_Policy > & *y*) [inline]

Divide and assign operator.

11.16.3.23 template<typename T , typename Policy > template<typename From > Checked_Number< T, Policy > & Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator= (const From & *y*) [inline]

Assignment operator.

Definition at line 497 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::assign_r().

11.16.3.24 template<typename T , typename Policy > Checked_Number< T, Policy > & Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator= (const Checked_Number< T, Policy > & *y*) [inline]

Assignment operator.

Definition at line 490 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.3.25 template<typename T , typename Policy > const T & Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value () const [inline]

Returns a const reference to the underlying numeric value.

Definition at line 272 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

11.16.3.26 template<typename T , typename Policy > T & Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value () [inline]

Returns a reference to the underlying numeric value.

Definition at line 266 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::v.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::external_memory_in_bytes(), Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::is_not_a_number(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator=(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >::raw_value(), Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >::raw_value(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::swap(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::total_memory_in_bytes().

11.16.4 Friends And Related Function Documentation

11.16.4.1 template<typename T , typename Policy > void abs_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x the absolute value of y.

11.16.4.2 template<typename T , typename Policy > void abs_assign (Checked_Number< T, Policy > & *x*) [related]

Assigns to x its absolute value.

11.16.4.3 template<typename T , typename Policy > void add_mul_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Assigns to x the value x + y * z.

11.16.4.4 template<typename To , typename To_Policy > Enable_If< Is_Native_Or_Checked< To >::value, Result >::type assign_r (To & *to*, char * *x*, Rounding_Dir *dir*) [related]

11.16.4.5 template<typename To > Enable_If< Is_Native_Or_Checked< To >::value, Result >::type assign_r (To & *to*, const char * *x*, Rounding_Dir *dir*) [related]

Definition at line 338 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::raw_value(), and Parma_Polyhedra_Library::rounding_dir().

11.16.4.6 template<typename To , typename From > Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type assign_r (To & *to*, const From & *x*, Rounding_Dir *dir*) [related]

Definition at line 205 of file Checked_Number.inlines.hh.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator=().

11.16.4.7 template<typename T , typename Policy > void ceil_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x smallest integral value not less than y.

11.16.4.8 template<typename T , typename Policy > void ceil_assign (Checked_Number< T, Policy > & *x*) [related]

Assigns to x smallest integral value not less than x.

11.16.4.9 template<typename From1 , typename From2 > Enable_If< Is_Native_Or_Checked< From1 >::value &&Is_Native_Or_Checked< From2 >::value, int >::type cmp (const From1 & *x*, const From2 & *y*) [related]

Returns a negative, zero or positive value depending on whether x is lower than, equal to or greater than y, respectively.

Definition at line 738 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::raw_value(), and Parma_Polyhedra_Library::Checked::Result_-Relation.

11.16.4.10 template<typename To , typename From > Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type construct (To & *to*, const From & *x*, Rounding_Dir *dir*) [related]

Definition at line 215 of file Checked_Number.inlines.hh.

11.16.4.11 template<typename T , typename Policy > void div_2exp_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, unsigned int *exp*) [related]

Assigns to x the value $y/2^{\mathtt{exp}}$.

11.16.4.12 template<typename T , typename Policy > void exact_div_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

If z divides y, assigns to x the quotient of the integer division of y and z.

The behavior is undefined if z does not divide y.

Definition at line 710 of file Checked_Number.inlines.hh.

11.16.4.13 template<typename T , typename Policy > memory_size_type external_memory_in_bytes (const Checked_Number< T, Policy > & *x*) [related]

Returns the size in bytes of the memory managed by x.

Definition at line 330 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::external_memory_in_bytes(), and Parma_Polyhedra_-Library::Checked_Number< T, Policy >::raw_value().

11.16.4.14 template<typename T , typename Policy > void floor_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x largest integral value not greater than y.

11.16.4.15 template<typename T , typename Policy > void floor_assign (Checked_Number< T, Policy > & *x*) [related]

Assigns to x largest integral value not greater than x.

11.16.4.16 template<typename T , typename Policy > void gcd_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Assigns to x the greatest common divisor of y and z.

11.16.4.17 template<typename T , typename Policy > void gcdext_assign (Checked_Number< T, Policy > & *x*, Checked_Number< T, Policy > & *s*, Checked_Number< T, Policy > & *t*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Assigns to x the greatest common divisor of y and z, setting s and t such that s∗y + t∗z = x = gcd(y, z).

11.16.4.18 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, Result >::type input (T & *x*, std::istream & *is*, Rounding_Dir *dir*) [related]

Input function.

Parameters

is Input stream to read from;

x Number (possibly extended) to assign to in case of successful reading;

dir Rounding mode to be applied.

Returns

Result of the input operation. Success, success with imprecision, overflow, parsing error: all possibilities are taken into account, checked for, and properly reported.

This function attempts reading a (possibly extended) number from the given stream is, possibly rounding as specified by dir, assigning the result to x upon success, and returning the appropriate Result.

The input syntax allows the specification of:

- plain base-10 integer numbers as 34976098, -77 and +13;
- base-10 integer numbers in scientific notation as 15e2 and 15*^2 (both meaning $15 \cdot 10^2 = 1500$), 9200e-2 and -18*^+1111111111111111;
- base-10 rational numbers in fraction notation as 15/3 and 15/-3;
- base-10 rational numbers in fraction/scientific notation as 15/30e-1 (meaning 5) and 15*^-3/29e2 (meaning $3/580000$);

- base-10 rational numbers in floating point notation as `71.3` (meaning $713/10$) and `-0.123456` (meaning $-1929/15625$);
- base-10 rational numbers in floating point scientific notation as `2.2e-1` (meaning $11/50$) and `-2.20001*^+3` (meaning $-220001/100$);
- integers and rationals (in fractional, floating point and scientific notations) specified by using Mathematica-style bases, in the range from 2 to 36, as `2^^11` (meaning 3), `36^^z` (meaning 35), `36^^xyz` (meaning 44027), `2^^11.1` (meaning $7/2$), `10^^2e3` (meaning 2000), `8^^2e3` (meaning 1024), `8^^2.1e3` (meaning 1088), `8^^20402543.120347e7` (meaning 9073863231288), `8^^2.1` (meaning $17/8$); note that the base and the exponent are always written as plain base-10 integer numbers; also, when an ambiguity may arise, the character `e` is interpreted as a digit, so that `16^^1e2` (meaning 482) is different from `16^^1*^2` (meaning 256);
- the C-style hexadecimal prefix `0x` is interpreted as the Mathematica-style prefix `16^^`;
- the C-style binary exponent indicator `p` can only be used when base 16 has been specified; if used, the exponent will be applied to base 2 (instead of base 16, as is the case when the indicator `e` is used);
- special values like `inf` and `+inf` (meaning $+\infty$), `-inf` (meaning $-\infty$), and `nan` (meaning "not a number").

The rationale behind the accepted syntax can be summarized as follows:

- if the syntax is accepted by Mathematica, then this function accepts it with the same semantics;
- if the syntax is acceptable as standard C++ integer or floating point literal (except for octal notation and type suffixes, which are not supported), then this function accepts it with the same semantics;
- natural extensions of the above are accepted with the natural extensions of the semantics;
- special values are accepted.

Valid syntax is more formally and completely specified by the following grammar, with the additional provisos that everything is *case insensitive*, that the syntactic category `BDIGIT` is further restricted by the current base and that for all bases above 14, any `e` is always interpreted as a digit and never as a delimiter for the exponent part (if such a delimiter is desired, it has to be written as `*^`).

```
number  : NAN
        | SIGN INF
        | INF
        | num
        | num DIV num
        ;

num     : unum
        | SIGN unum

unum    : unum1
        | HEX unum1
        | base BASE unum1
        ;

unum1   : mantissa
        | mantissa EXP exponent
        ;

mantissa: bdigits
        | POINT bdigits
        | bdigits POINT
        | bdigits POINT bdigits
```

```
INF     : 'inf'
        ;

NAN     : 'nan'
        ;

SIGN    : '-'
        | '+'
        ;

EXP     : 'e'
        | 'p'
        | '*^'
        ;
POINT   : '.'
        ;

DIV     : '/'
        ;

MINUS   : '-'
        ;
```

```
        ;

exponent: SIGN digits
        | digits
        ;

bdigits : BDIGIT
        | bdigits BDIGIT
        ;

digits  : DIGIT
        | digits DIGIT
        ;

PLUS    : '+'
        ;

HEX     : '0x'
        ;

BASE    : '^^'
        ;

DIGIT   : '0' .. '9'
        ;

BDIGIT  : '0' .. '9'
        | 'a' .. 'z'
        ;
```

Definition at line 780 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::raw_value(), and Parma_Polyhedra_Library::rounding_dir().

11.16.4.19 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, int >::type is_infinity (const T & *x*) [related]

Definition at line 239 of file Checked_Number.inlines.hh.

11.16.4.20 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_integer (const T & *x*) [related]

Definition at line 252 of file Checked_Number.inlines.hh.

11.16.4.21 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_minus_infinity (const T & *x*) [related]

Definition at line 225 of file Checked_Number.inlines.hh.

11.16.4.22 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_not_a_number (const T & *x*) [related]

Definition at line 245 of file Checked_Number.inlines.hh.

11.16.4.23 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, bool >::type is_plus_infinity (const T & *x*) [related]

Definition at line 232 of file Checked_Number.inlines.hh.

11.16.4.24 template<typename T , typename Policy > void lcm_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Assigns to x the least common multiple of y and z.

11.16.4.25 template<typename T , typename Policy > void mul_2exp_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, unsigned int *exp*) [related]

Assigns to x the value $y \cdot 2^{\mathtt{exp}}$.

11.16.4.26 template<typename T , typename Policy > void neg_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x the negation of y.

11.16.4.27 template<typename T , typename Policy > void neg_assign (Checked_Number< T, Policy > & *x*) [related]

Assigns to x its negation.

11.16.4.28 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator!= (const T1 & *x*, const T2 & *y*) [related]

Disequality operator.

Definition at line 183 of file Interval.inlines.hh.

11.16.4.29 template<typename T , typename Policy > Checked_Number< T, Policy > operator+ (const Checked_Number< T, Policy > & *x*) [related]

Unary plus operator.

Definition at line 617 of file Checked_Number.inlines.hh.

11.16.4.30 template<typename T , typename Policy > Checked_Number< T, Policy > operator- (const Checked_Number< T, Policy > & *x*) [related]

Unary minus operator.

Definition at line 624 of file Checked_Number.inlines.hh.

11.16.4.31 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator< (const T1 & *x*, const T2 & *y*) [related]

Less than operator.

11.16.4.32 template<typename T , typename Policy > std::ostream & operator<< (std::ostream & *os*, const Checked_Number< T, Policy > & *x*) [related]

Output operator.

Definition at line 772 of file Checked_Number.inlines.hh.

11.16.4.33 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator<= (const T1 & *x*, const T2 & *y*) [related]

Less than or equal to operator.

11.16.4.34 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator== (const T1 & *x*, const T2 & *y*) [related]

Equality operator.

Definition at line 164 of file Interval.inlines.hh.

11.16.4.35 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator> (const T1 & *x*, const T2 & *y*) [related]

Greater than operator.

11.16.4.36 template<typename T1 , typename T2 > Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value &&(Is_Checked< T1 >::value||Is_Checked< T2 >::value), bool >::type operator>= (const T1 & *x*, const T2 & *y*) [related]

Greater than or equal to operator.

11.16.4.37 template<typename T , typename Policy > std::istream & operator>> (std::istream & *is*, Checked_Number< T, Policy > & *x*) [related]

Input operator.

Definition at line 790 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked::Result.

11.16.4.38 template<typename T > Enable_If< Is_Native_Or_Checked< T >::value, Result >::type output (std::ostream & *os*, const T & *x*, const Numeric_Format & *fmt*, Rounding_Dir *dir*) [related]

Definition at line 759 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::check_result(), Parma_Polyhedra_Library::raw_value(), and Parma_Polyhedra_Library::rounding_dir().

11.16.4.39 template<typename T , typename Policy > T & raw_value (Checked_Number< T, Policy > & *x*) [related]

Definition at line 286 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

11.16.4.40 template<typename T , typename Policy > const T & raw_value (const Checked_Number< T, Policy > & *x*) [related]

Definition at line 279 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

11.16.4.41 template<typename From > Enable_If< Is_Native_Or_Checked< From >::value, int >::type sgn (const From & *x*) [related]

Returns -1, 0 or 1 depending on whether the value of x is negative, zero or positive, respectively.

Definition at line 719 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::raw_value(), and Parma_Polyhedra_Library::Checked::Result_-Relation.

11.16.4.42 template<typename T , typename Policy > void sqrt_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x the integer square root of y.

11.16.4.43 template<typename T , typename Policy > void sub_mul_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*, const Checked_Number< T, Policy > & *z*) [related]

Assigns to x the value x - y * z.

11.16.4.44 template<typename T , typename Policy > void swap (Checked_Number< T, Policy > & *x*, Checked_Number< T, Policy > & *y*) [related]

Swaps x with y.

Definition at line 821 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_-Polyhedra_Library::swap().

11.16.4.45 template<typename T , typename Policy > memory_size_type total_memory_in_bytes (const Checked_Number< T, Policy > & *x*) [related]

Returns the total size in bytes of the memory occupied by x.

Definition at line 323 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_-Polyhedra_Library::total_memory_in_bytes().

11.16.4.46 template<typename T , typename Policy > void trunc_assign (Checked_Number< T, Policy > & *x*, const Checked_Number< T, Policy > & *y*) [related]

Assigns to x the value of y rounded to the nearest integer not larger in absolute value.

11.16.4.47 template<typename T , typename Policy > void trunc_assign (Checked_Number< T, Policy > & *x*) [related]

Round x to the nearest integer not larger in absolute value.

11.16.5 Member Data Documentation

11.16.5.1 template<typename T, typename Policy> T Parma_Polyhedra_Library::Checked_Number< T, Policy >::v [private]

The underlying numeric value.

Definition at line 424 of file Checked_Number.defs.hh.

Referenced by Parma_Polyhedra_Library::Checked_Number< T, Policy >::Checked_Number(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::classify(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator T(), Parma_Polyhedra_Library::Checked_Number< T, Policy >::operator=(), and Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value().

The documentation for this class was generated from the following files:

- Checked_Number.defs.hh
- Checked_Number.inlines.hh

11.17 Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T > Struct Template Reference

`#include <checked.defs.hh>`

Public Member Functions

- const_bool_nodef (check_overflow, false)
  Do not check for overflowed result.
- const_bool_nodef (check_inf_add_inf, false)
  Do not check for attempts to add infinities with different sign.
- const_bool_nodef (check_inf_sub_inf, false)
  Do not check for attempts to subtract infinities with same sign.
- const_bool_nodef (check_inf_mul_zero, false)
  Do not check for attempts to multiply infinities by zero.
- const_bool_nodef (check_div_zero, false)
  Do not check for attempts to divide by zero.
- const_bool_nodef (check_inf_div_inf, false)
  Do not check for attempts to divide infinities.
- const_bool_nodef (check_inf_mod, false)
  Do not check for attempts to compute remainder of infinities.
- const_bool_nodef (check_sqrt_neg, false)

Do not check for attempts to take the square root of a negative number.

- const_bool_nodef (has_nan, std::numeric_limits< T >::has_quiet_NaN)

  Handle not-a-number special value if `T` has it.

- const_bool_nodef (has_infinity, std::numeric_limits< T >::has_infinity)

  Handle infinity special values if `T` have them.

- const_bool_nodef (convertible, true)

  The checked number can always be safely converted to the underlying type `T` and vice-versa.

- const_bool_nodef (fpu_check_inexact, false)

  Do not honor requests to check for FPU inexact results.

- const_bool_nodef (fpu_check_nan_result, false)

  Do not make extra checks to detect FPU NaN results.

Static Public Member Functions

- static void handle_result (Result r)

  Handles `r`: called by all constructors, operators and functions that do not return a Result value.

Static Public Attributes

- static const Rounding_Dir ROUND_DEFAULT_CONSTRUCTOR = ROUND_NATIVE

  For constructors, by default use the same rounding used by underlying type.

- static const Rounding_Dir ROUND_DEFAULT_OPERATOR = ROUND_NATIVE

  For overloaded operators (operator+(), operator-(), ...), by default use the same rounding used by the underlying type.

- static const Rounding_Dir ROUND_DEFAULT_INPUT = ROUND_NATIVE

  For input functions, by default use the same rounding used by the underlying type.

- static const Rounding_Dir ROUND_DEFAULT_OUTPUT = ROUND_NATIVE

  For output functions, by default use the same rounding used by the underlying type.

- static const Rounding_Dir ROUND_DEFAULT_FUNCTION = ROUND_NATIVE

  For all other functions, by default use the same rounding used by the underlying type.

11.17.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >

Definition at line 570 of file checked.defs.hh.

11.17.2 Member Function Documentation

11.17.2.1 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (fpu_check_nan_result, false)

Do not make extra checks to detect FPU NaN results.

11.17.2.2 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (fpu_check_inexact, false)

Do not honor requests to check for FPU inexact results.

11.17.2.3 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (convertible, true)

The checked number can always be safely converted to the underlying type `T` and vice-versa.

11.17.2.4 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (has_infinity, std::numeric_limits< T >::has_infinity)

Handle infinity special values if `T` have them.

11.17.2.5 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (has_nan, std::numeric_limits< T >::has_quiet_NaN)

Handle not-a-number special value if `T` has it.

11.17.2.6 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_sqrt_neg, false)

Do not check for attempts to take the square root of a negative number.

11.17.2.7 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_inf_mod, false)

Do not check for attempts to compute remainder of infinities.

11.17.2.8 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_inf_div_inf, false)

Do not check for attempts to divide infinities.

11.17.2.9 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_div_zero, false)

Do not check for attempts to divide by zero.

11.17.2.10 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_inf_mul_zero, false)

Do not check for attempts to multiply infinities by zero.

11.17.2.11 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_inf_sub_inf, false)

Do not check for attempts to subtract infinities with same sign.

11.17.2.12 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_inf_add_inf, false)

Do not check for attempts to add infinities with different sign.

11.17.2.13 template<typename T > Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::const_bool_nodef (check_overflow, false)

Do not check for overflowed result.

11.17.2.14 template<typename T > void Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::handle_result (Result *r*) [inline, static]

Handles `r`: called by all constructors, operators and functions that do not return a Result value.

Definition at line 61 of file Checked_Number.inlines.hh.

11.17.3 Member Data Documentation

11.17.3.1 template<typename T > const Rounding_Dir Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::ROUND_DEFAULT_CONSTRUCTOR = ROUND_NATIVE [static]

For constructors, by default use the same rounding used by underlying type.

Definition at line 617 of file checked.defs.hh.

11.17.3.2 template<typename T > const Rounding_Dir Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::ROUND_DEFAULT_FUNCTION = ROUND_NATIVE [static]

For all other functions, by default use the same rounding used by the underlying type.

Definition at line 641 of file checked.defs.hh.

11.17.3.3 template<typename T > const Rounding_Dir Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::ROUND_DEFAULT_INPUT = ROUND_NATIVE [static]

For input functions, by default use the same rounding used by the underlying type.

Definition at line 629 of file checked.defs.hh.

11.17.3.4 template<typename T > const Rounding_Dir Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::ROUND_DEFAULT_OPERATOR = ROUND_NATIVE [static]

For overloaded operators (operator+(), operator-(), ...), by default use the same rounding used by the underlying type.

Definition at line 623 of file checked.defs.hh.

11.17.3.5 template<typename T > const Rounding_Dir Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >::ROUND_DEFAULT_OUTPUT = ROUND_NATIVE [static]

For output functions, by default use the same rounding used by the underlying type.

Definition at line 635 of file checked.defs.hh.

The documentation for this struct was generated from the following files:

- checked.defs.hh
- Checked_Number.inlines.hh

11.18 Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient > Struct Template Reference

Coefficient traits.

`#include <Coefficient_traits_template.hh>`

11.18.1 Detailed Description

template<typename Coefficient> struct Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient >

Coefficient traits.

Definition at line 23 of file Coefficient_traits_template.hh.

The documentation for this struct was generated from the following file:

- Coefficient_traits_template.hh

11.19 Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer > Struct Template Reference

Coefficient traits specialization for unbounded integers.

`#include <GMP_Integer.types.hh>`

Public Types

- typedef const GMP_Integer & const_reference

  The type used for references to const unbounded integers.

11.19.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer >

Coefficient traits specialization for unbounded integers.

Definition at line 38 of file GMP_Integer.types.hh.

11.19.2 Member Typedef Documentation

11.19.2.1 typedef const GMP_Integer& Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer >::const_reference

The type used for references to const unbounded integers.

Definition at line 40 of file GMP_Integer.types.hh.

The documentation for this struct was generated from the following file:

- GMP_Integer.types.hh

11.20 Parma_Polyhedra_Library::BHRZ03_Certificate::Compare Struct Reference

A total ordering on BHRZ03 certificates.

`#include <BHRZ03_Certificate.defs.hh>`

Public Member Functions

- bool operator() (const BHRZ03_Certificate &x, const BHRZ03_Certificate &y) const

  Returns `true` *if and only if* `x` *comes before* `y`.

11.20.1 Detailed Description

A total ordering on BHRZ03 certificates. This binary predicate defines a total ordering on BHRZ03 certificates which is used when storing information about sets of polyhedra.

Definition at line 85 of file BHRZ03_Certificate.defs.hh.

11.20.2 Member Function Documentation

11.20.2.1 bool Parma_Polyhedra_Library::BHRZ03_Certificate::Compare::operator() (const BHRZ03_Certificate & *x*, const BHRZ03_Certificate & *y*) const [inline]

Returns `true` if and only if `x` comes before `y`.

Definition at line 54 of file BHRZ03_Certificate.inlines.hh.

References Parma_Polyhedra_Library::BHRZ03_Certificate::compare().

The documentation for this struct was generated from the following files:

- BHRZ03_Certificate.defs.hh
- BHRZ03_Certificate.inlines.hh

11.21 Parma_Polyhedra_Library::H79_Certificate::Compare Struct Reference

A total ordering on H79 certificates.

`#include <H79_Certificate.defs.hh>`

Public Member Functions

- bool operator() (const H79_Certificate &x, const H79_Certificate &y) const

  Returns `true` *if and only if* `x` *comes before* `y`.

11.21.1 Detailed Description

A total ordering on H79 certificates. This binary predicate defines a total ordering on H79 certificates which is used when storing information about sets of polyhedra.

Definition at line 83 of file H79_Certificate.defs.hh.

11.21.2 Member Function Documentation

11.21.2.1 bool Parma_Polyhedra_Library::H79_Certificate::Compare::operator() (const H79_Certificate & *x*, const H79_Certificate & *y*) const [inline]

Returns `true` if and only if `x` comes before `y`.

Definition at line 47 of file H79_Certificate.inlines.hh.

References Parma_Polyhedra_Library::H79_Certificate::compare().

The documentation for this struct was generated from the following files:

- H79_Certificate.defs.hh
- H79_Certificate.inlines.hh

11.22 Parma_Polyhedra_Library::Grid_Certificate::Compare Struct Reference

A total ordering on Grid certificates.

`#include <Grid_Certificate.defs.hh>`

Public Member Functions

- bool operator() (const Grid_Certificate &x, const Grid_Certificate &y) const

  Returns `true` *if and only if* `x` *comes before* `y`.

11.22.1 Detailed Description

A total ordering on Grid certificates. This binary predicate defines a total ordering on Grid certificates which is used when storing information about sets of grids.

Definition at line 82 of file Grid_Certificate.defs.hh.

11.22.2 Member Function Documentation

11.22.2.1 bool Parma_Polyhedra_Library::Grid_Certificate::Compare::operator() (const Grid_Certificate & *x*, const Grid_Certificate & *y*) const [inline]

Returns `true` if and only if `x` comes before `y`.

Definition at line 52 of file Grid_Certificate.inlines.hh.

References Parma_Polyhedra_Library::Grid_Certificate::compare().

The documentation for this struct was generated from the following files:

- Grid_Certificate.defs.hh
- Grid_Certificate.inlines.hh

11.23 Parma_Polyhedra_Library::Variable::Compare Struct Reference

Binary predicate defining the total ordering on variables.

`#include <Variable.defs.hh>`

Public Member Functions

- bool operator() (Variable x, Variable y) const

  Returns `true` if and only if `x` comes before `y`.

11.23.1 Detailed Description

Binary predicate defining the total ordering on variables.

Definition at line 124 of file Variable.defs.hh.

11.23.2 Member Function Documentation

11.23.2.1 bool Parma_Polyhedra_Library::Variable::Compare::operator() (Variable *x*, Variable *y*) const [inline]

Returns `true` if and only if `x` comes before `y`.

Definition at line 83 of file Variable.inlines.hh.

References Parma_Polyhedra_Library::Variable::less().

The documentation for this struct was generated from the following files:

- Variable.defs.hh
- Variable.inlines.hh

11.24 Parma_Polyhedra_Library::Compile_Time_Check< true > Struct Template Reference

A class that is only defined if `b` evaluates to `true`.

`#include <meta_programming.hh>`

11.24.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Compile_Time_Check< true >

A class that is only defined if `b` evaluates to `true`. This is the specialized case with `b` equal to `true`, so the class is declared and (trivially) defined.

Definition at line 115 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

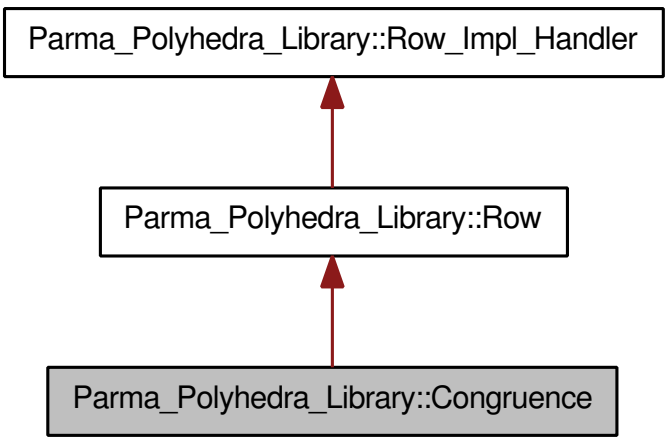
- meta_programming.hh

11.25 Parma_Polyhedra_Library::Congruence Class Reference

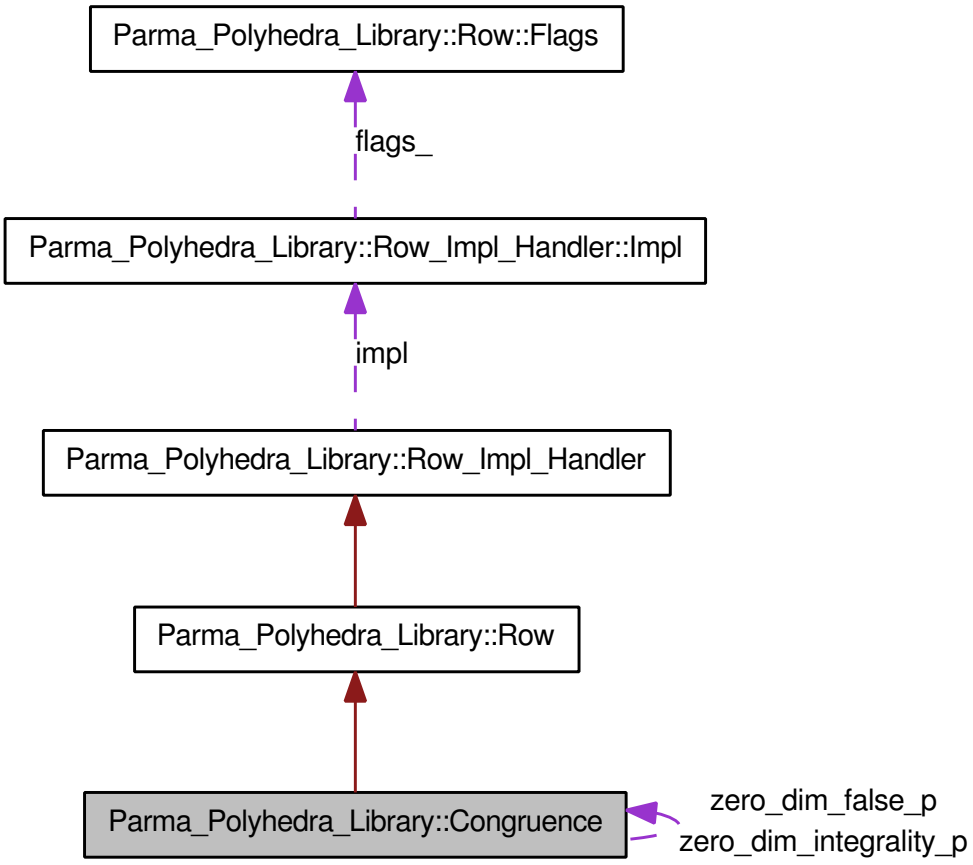
A linear congruence.

`#include <Congruence.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Congruence:



Collaboration diagram for Parma_Polyhedra_Library::Congruence:



Public Member Functions

- Congruence (const Congruence &cg)

  Ordinary copy constructor.

- Congruence (const Constraint &c)

  Copy-constructs (modulo 0) from equality constraint `c`.

- ~Congruence ()

  Destructor.

- Congruence & operator= (const Congruence &cg)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Coefficient_traits::const_reference coefficient (Variable v) const

  Returns the coefficient of `v` *in* `*this`.

- Coefficient_traits::const_reference inhomogeneous_term () const

  Returns the inhomogeneous term of `*this`.

- Coefficient_traits::const_reference modulus () const

  Returns a const reference to the modulus of `*this`.

- Congruence & operator/= (Coefficient_traits::const_reference k)

  Multiplies `k` *into the modulus of* `*this`.

- bool is_tautological () const

  Returns `true` *if and only if* `*this` *is a tautology (i.e., an always true congruence).*

- bool is_inconsistent () const

  Returns `true` *if and only if* `*this` *is inconsistent (i.e., an always false congruence).*

- bool is_proper_congruence () const

  Returns `true` *if the modulus is greater than zero.*

- bool is_equality () const

  Returns `true` *if* `*this` *is an equality.*

- bool is_equal_at_dimension (dimension_type dim, const Congruence &cg) const

  Returns `true` *if* `*this` *is equal to* `cg` *in dimension* `dim`.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation of the internal representation of* `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Congruence can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Congruence & zero_dim_integrality ()

  Returns a reference to the true (zero-dimension space) congruence $0 = 1 \pmod 1$*, also known as the integrality congruence.*

- static const Congruence & zero_dim_false ()

  Returns a reference to the false (zero-dimension space) congruence $0 = 1 \pmod 0$.

- static Congruence create (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the congruence $e1 = e2 \pmod 1$.

- static Congruence create (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the congruence $e = n \pmod 1$.

- static Congruence create (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the congruence $n = e \pmod 1$.

Protected Member Functions

- void sign_normalize ()

  Normalizes the signs.

- void normalize ()

  Normalizes signs and the inhomogeneous term.

- void strong_normalize ()

  Calls normalize, then divides out common factors.

Private Member Functions

- void set_is_equality ()

  Marks this congruence as a linear equality.

- void negate (dimension_type start, dimension_type end)

  Negates the elements from index `start` *to index* `end`.

- Congruence ()

  Default constructor: private and not implemented.

- Congruence (const Congruence &cg, dimension_type sz, dimension_type capacity)

  Copy-constructs with specified size and capacity.

- Congruence (const Constraint &c, dimension_type sz, dimension_type capacity)

  Constructs from a constraint, with specified size and capacity.

- Congruence (const Congruence &cg, Coefficient_traits::const_reference k)

  Copy-constructs from `cg`*, multiplying* `k` *into the modulus.*

- Congruence (Linear_Expression &le, Coefficient_traits::const_reference m)

  Constructs from Linear_Expression `le`*, using modulus* `m`*.*

- void swap (Congruence &y)

  Swaps `*this` *with* `y`*.*

- void throw_invalid_argument (const char *method, const char *message) const

  Throws a `std::invalid_argument` *exception containing error message* `message`*.*

- void throw_dimension_incompatible (const char *method, const char *v_name, Variable v) const

  Throws a `std::invalid_argument` *exception containing the appropriate error message.*

Static Private Attributes

- static const Congruence * zero_dim_false_p = 0

  Holds (between class initialization and finalization) a pointer to the false (zero-dimension space) congruence $0 = 1 \pmod 0$.

- static const Congruence * zero_dim_integrality_p = 0

  Holds (between class initialization and finalization) a pointer to the true (zero-dimension space) congruence $0 = 1 \pmod 1$*, also known as the integrality congruence.*

Friends

- class Parma_Polyhedra_Library::Scalar_Products
- class Parma_Polyhedra_Library::Constraint
- class Parma_Polyhedra_Library::Congruence_System
- class Parma_Polyhedra_Library::Congruence_System::const_iterator
- class Parma_Polyhedra_Library::Grid
- class Parma_Polyhedra_Library::Linear_Expression
- Congruence operator/ (const Congruence &cg, Coefficient_traits::const_reference k)

  Returns a copy of `cg`*, multiplying* `k` *into the copy's modulus.*

- Congruence operator/ (const Constraint &c, Coefficient_traits::const_reference m)

  Creates a congruence from `c`*, with* `m` *as the modulus.*

- bool operator== (const Congruence &x, const Congruence &y)

  Returns `true` *if and only if* `x` *and* `y` *are equivalent.*

- bool operator!= (const Congruence &x, const Congruence &y)

  Returns `false` *if and only if* `x` *and* `y` *are equivalent.*

- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Congruence_System &cgs)
- void std::swap (Parma_Polyhedra_Library::Congruence &x, Parma_Polyhedra_Library::Congruence &y)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Congruence &c)

  Output operators.

- Congruence operator%= (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the congruence $e1 = e2 \pmod 1$.

- Congruence operator%= (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the congruence $e = n \pmod 1$.

- void swap (Parma_Polyhedra_Library::Congruence &x, Parma_Polyhedra_Library::Congruence &y)

  Specializes `std::swap`.

11.25.1 Detailed Description

A linear congruence. An object of the class Congruence is a congruence:

- $\mathrm{cg} = \sum_{i=0}^{n-1} a_i x_i + b = 0 \pmod m$

where n is the dimension of the space, a_i is the integer coefficient of variable x_i, b is the integer inhomogeneous term and m is the integer modulus; if $m = 0$, then cg represents the equality congruence $\sum_{i=0}^{n-1} a_i x_i + b = 0$ and, if $m \neq 0$, then the congruence cg is said to be a proper congruence.

How to build a congruence

Congruences $\pmod 1$ are typically built by applying the congruence symbol '`%=`' to a pair of linear expressions. Congruences with modulus `m` are typically constructed by building a congruence $\pmod 1$ using the given pair of linear expressions and then adding the modulus `m` using the modulus symbol is '`/`'.

The space dimension of a congruence is defined as the maximum space dimension of the arguments of its constructor.

In the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds the equality congruence $3x + 5y - z = 0$, having space dimension 3:

```
Congruence eq_cg((3*x + 5*y - z %= 0) / 0);
```

The following code builds the congruence $4x = 2y - 13 \pmod 1$, having space dimension 2:

```
Congruence mod1_cg(4*x %= 2*y - 13);
```

The following code builds the congruence $4x = 2y - 13 \pmod 2$, having space dimension 2:

```
Congruence mod2_cg((4*x %= 2*y - 13) / 2);
```

An unsatisfiable congruence on the zero-dimension space $\mathbb{R}^0$ can be specified as follows:

```
Congruence false_cg = Congruence::zero_dim_false();
```

Equivalent, but more involved ways are the following:

```
Congruence false_cg1((Linear_Expression::zero() %= 1) / 0);
Congruence false_cg2((Linear_Expression::zero() %= 1) / 2);
```

In contrast, the following code defines an unsatisfiable congruence having space dimension 3:

```
Congruence false_cg3((0*z %= 1) / 0);
```

How to inspect a congruence

Several methods are provided to examine a congruence and extract all the encoded information: its space dimension, its modulus and the value of its integer coefficients.

Example 2

The following code shows how it is possible to access the modulus as well as each of the coefficients. Given a congruence with linear expression `e` and modulus `m` (in this case $x - 5y + 3z = 4 \pmod 5$), we construct a new congruence with the same modulus `m` but where the linear expression is $2e$ ($2x - 10y + 6z = 8 \pmod 5$).

```
Congruence cg1((x - 5*y + 3*z %= 4) / 5);
cout << "Congruence cg1: " << cg1 << endl;
const Coefficient& m = cg1.modulus();
if (m == 0)
  cout << "Congruence cg1 is an equality." << endl;
else {
  Linear_Expression e;
  for (dimension_type i = cg1.space_dimension(); i-- > 0; )
    e += 2 * cg1.coefficient(Variable(i)) * Variable(i);
    e += 2 * cg1.inhomogeneous_term();
  Congruence cg2((e %= 0) / m);
  cout << "Congruence cg2: " << cg2 << endl;
}
```

The actual output could be the following:

```
Congruence cg1: A - 5*B + 3*C %= 4 / 5
Congruence cg2: 2*A - 10*B + 6*C %= 8 / 5
```

Note that, in general, the particular output obtained can be syntactically different from the (semantically equivalent) congruence considered.

Definition at line 211 of file Congruence.defs.hh.

11.25.2 Constructor & Destructor Documentation

11.25.2.1 Parma_Polyhedra_Library::Congruence::Congruence (const Congruence & *cg*) [inline]

Ordinary copy constructor.

Definition at line 35 of file Congruence.inlines.hh.

11.25.2.2 Parma_Polyhedra_Library::Congruence::Congruence (const Constraint & *c*) [explicit]

Copy-constructs (modulo 0) from equality constraint c.

Exceptions

std::invalid_argument Thrown if c is an inequality.

Definition at line 37 of file Congruence.cc.

References Parma_Polyhedra_Library::Row::size().

11.25.2.3 Parma_Polyhedra_Library::Congruence::~Congruence () [inline]

Destructor.

Definition at line 56 of file Congruence.inlines.hh.

11.25.2.4 Parma_Polyhedra_Library::Congruence::Congruence () [private]

Default constructor: private and not implemented.

Referenced by initialize().

11.25.2.5 Parma_Polyhedra_Library::Congruence::Congruence (const Congruence & *cg*, dimension_type *sz*, dimension_type *capacity*) [inline, private]

Copy-constructs with specified size and capacity.

Definition at line 40 of file Congruence.inlines.hh.

11.25.2.6 Parma_Polyhedra_Library::Congruence::Congruence (const Constraint & *c*, dimension_type *sz*, dimension_type *capacity*) [private]

Constructs from a constraint, with specified size and capacity.

Definition at line 48 of file Congruence.cc.

11.25.2.7 Parma_Polyhedra_Library::Congruence::Congruence (const Congruence & *cg*, Coefficient_traits::const_reference *k*) [inline, private]

Copy-constructs from `cg`, multiplying `k` into the modulus.

If `cg` represents the congruence $e_1 = e_2 \pmod{m}$, then the result represents the congruence $e_1 = e_2 \pmod{mk}$.

Definition at line 46 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

11.25.2.8 Parma_Polyhedra_Library::Congruence::Congruence (Linear_Expression & *le*, Coefficient_traits::const_reference *m*) [inline, private]

Constructs from Linear_Expression `le`, using modulus `m`.

Builds a congruence with modulus `m`, stealing the coefficients from `le`.

Parameters

le The Linear_Expression holding the coefficients.

m The modulus for the congruence, which must be zero or greater.

Definition at line 60 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size(), and swap().

11.25.3 Member Function Documentation

11.25.3.1 void Parma_Polyhedra_Library::Congruence::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 208 of file Congruence.cc.

References Parma_Polyhedra_Library::Row::size().

11.25.3.2 void Parma_Polyhedra_Library::Congruence::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

11.25.3.3 bool Parma_Polyhedra_Library::Congruence::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation of the internal representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 223 of file Congruence.cc.

References Parma_Polyhedra_Library::Row::shrink(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Row::swap().

11.25.3.4 Coefficient_traits::const_reference Parma_Polyhedra_Library::Congruence::coefficient (Variable *v*) const [inline]

Returns the coefficient of `v` in `*this`.

Exceptions

std::invalid_argument thrown if the index of `v` is greater than or equal to the space dimension of `*this`.

Definition at line 167 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), Parma_Polyhedra_Library::extract_interval_congruence(), operator<<(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::shrink_to_congruence_no_check().

11.25.3.5 Congruence Parma_Polyhedra_Library::Congruence::create (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [inline, static]

Returns the congruence $n = e \pmod 1$.

Definition at line 78 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.25.3.6 Congruence Parma_Polyhedra_Library::Congruence::create (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [inline, static]

Returns the congruence $e = n \pmod 1$.

Definition at line 68 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.25.3.7 PPL::Congruence Parma_Polyhedra_Library::Congruence::create (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [static]

Returns the congruence $e1 = e2 \pmod 1$.

Definition at line 111 of file Congruence.cc.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by operator%=().

11.25.3.8 memory_size_type Parma_Polyhedra_Library::Congruence::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 212 of file Congruence.inlines.hh.

11.25.3.9 void Parma_Polyhedra_Library::Congruence::finalize () [static]

Finalizes the class.

Definition at line 286 of file Congruence.cc.

References zero_dim_false_p, and zero_dim_integrality_p.

11.25.3.10 Coefficient_traits::const_reference Parma_Polyhedra_Library::Congruence::inhomogeneous_term () const [inline]

Returns the inhomogeneous term of `*this`.

Definition at line 174 of file Congruence.inlines.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), is_inconsistent(), is_tautological(), operator<<(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::shrink_to_congruence_no_check(), and Parma_Polyhedra_Library::Grid::simplify().

11.25.3.11 void Parma_Polyhedra_Library::Congruence::initialize () [static]

Initializes the class.

Definition at line 275 of file Congruence.cc.

References Congruence(), Parma_Polyhedra_Library::Linear_Expression::zero(), zero_dim_false_p, and zero_dim_integrality_p.

11.25.3.12 bool Parma_Polyhedra_Library::Congruence::is_equal_at_dimension (dimension_type *dim*, const Congruence & *cg*) const [inline]

Returns `true` if `*this` is equal to `cg` in dimension `dim`.

Definition at line 195 of file Congruence.inlines.hh.

References modulus(), and Parma_Polyhedra_Library::Row::operator[ ]().

Referenced by Parma_Polyhedra_Library::Grid::select_wider_congruences().

11.25.3.13 bool Parma_Polyhedra_Library::Congruence::is_equality () const [inline]

Returns `true` if `*this` is an equality.

A modulus of zero denotes a linear equality.

Definition at line 190 of file Congruence.inlines.hh.

References modulus().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::extract_interval_congruence(), is_tautological(), Parma_Polyhedra_Library::Grid::multiply_grid(), Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruence(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), Parma_Polyhedra_Library::Grid::simplify(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.25.3.14 bool Parma_Polyhedra_Library::Congruence::is_inconsistent () const

Returns `true` if and only if `*this` is inconsistent (i.e., an always false congruence).

An inconsistent congruence has one of the following two forms:

- an equality: $\sum_{i=0}^{n-1} 0x_i + b == 0$ where $b \neq 0$; or
- a proper congruence: $\sum_{i=0}^{n-1} 0x_i + b\% = 0/m$, where $b \neq 0 \pmod{m}$.

Definition at line 196 of file Congruence.cc.

References inhomogeneous_term(), is_proper_congruence(), modulus(), and space_dimension().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_-congruence(), Parma_Polyhedra_Library::Grid::add_congruence_no_check(), Parma_Polyhedra_-Library::Box< ITV >::add_congruence_no_check(), Parma_Polyhedra_Library::Polyhedron::add_-congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_-Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.25.3.15 bool Parma_Polyhedra_Library::Congruence::is_proper_congruence () const [inline]

Returns `true` if the modulus is greater than zero.

A congruence with a modulus of 0 is a linear equality.

Definition at line 185 of file Congruence.inlines.hh.

References modulus().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_-Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Grid::difference_-assign(), is_inconsistent(), is_tautological(), Parma_Polyhedra_Library::Grid::multiply_grid(), Parma_-Polyhedra_Library::Congruence_System::num_proper_congruences(), operator<<(), Parma_Polyhedra_-Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_-Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_-Polyhedra_Library::Grid::simplify(), and Parma_Polyhedra_Library::Grid::simplify_using_context_-assign().

11.25.3.16 bool Parma_Polyhedra_Library::Congruence::is_tautological () const

Returns `true` if and only if `*this` is a tautology (i.e., an always true congruence).

A tautological congruence has one the following two forms:

- an equality: $\sum_{i=0}^{n-1} 0x_i + 0 == 0$; or
- a proper congruence: $\sum_{i=0}^{n-1} 0x_i + b\% = 0/m$, where $b = 0 \pmod{m}$.

Definition at line 183 of file Congruence.cc.

References inhomogeneous_term(), is_equality(), is_proper_congruence(), modulus(), and space_-dimension().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), Parma_-Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Polyhedron::refine_-with_congruence(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.25.3.17 dimension_type Parma_Polyhedra_Library::Congruence::max_space_dimension () [inline, static]

Returns the maximum space dimension a Congruence can handle.

Definition at line 155 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::max_size().

11.25.3.18 Coefficient_traits::const_reference Parma_Polyhedra_Library::Congruence::modulus () const [inline]

Returns a const reference to the modulus of `*this`.

Definition at line 179 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Grid::difference_assign(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), is_equal_at_dimension(), is_equality(), is_inconsistent(), is_proper_congruence(), is_tautological(), normalize(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), OK(), operator<<(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Grid::reduce_equality_with_equality(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), Parma_Polyhedra_Library::shrink_to_congruence_no_check(), Parma_Polyhedra_Library::Grid::simplify(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.25.3.19 void Parma_Polyhedra_Library::Congruence::negate (dimension_type *start*, dimension_type *end*) [inline, private]

Negates the elements from index `start` to index `end`.

Definition at line 206 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::neg_assign().

Referenced by Parma_Polyhedra_Library::Grid::simplify().

11.25.3.20 void Parma_Polyhedra_Library::Congruence::normalize () [protected]

Normalizes signs and the inhomogeneous term.

Applies sign_normalize, then reduces the inhomogeneous term to the smallest possible positive number.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 84 of file Congruence.cc.

References modulus(), sign_normalize(), and Parma_Polyhedra_Library::Row::size().

Referenced by strong_normalize().

11.25.3.21 bool Parma_Polyhedra_Library::Congruence::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 253 of file Congruence.cc.

References modulus(), and Parma_Polyhedra_Library::Row::OK().

Referenced by Parma_Polyhedra_Library::Congruence_System::OK().

11.25.3.22 Congruence & Parma_Polyhedra_Library::Congruence::operator/= (Coefficient_traits::const_reference *k*) [inline]

Multiplies `k` into the modulus of `*this`.

If called with `*this` representing the congruence $e_1 = e_2 \pmod{m}$, then it returns with *this representing the congruence $e_1 = e_2 \pmod{mk}$.

Definition at line 130 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

11.25.3.23 Congruence & Parma_Polyhedra_Library::Congruence::operator= (const Congruence & *cg*) [inline]

Assignment operator.

Definition at line 117 of file Congruence.inlines.hh.

11.25.3.24 void Parma_Polyhedra_Library::Congruence::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Row.

11.25.3.25 void Parma_Polyhedra_Library::Congruence::set_is_equality () [inline, private]

Marks this congruence as a linear equality.

Definition at line 201 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Grid::simplify().

11.25.3.26 void Parma_Polyhedra_Library::Congruence::sign_normalize () [protected]

Normalizes the signs.

The signs of the coefficients and the inhomogeneous term are normalized, leaving the first non-zero homogeneous coefficient positive.

Definition at line 62 of file Congruence.cc.

References Parma_Polyhedra_Library::neg_assign(), and Parma_Polyhedra_Library::Row::size().

Referenced by normalize().

11.25.3.27 dimension_type Parma_Polyhedra_Library::Congruence::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 162 of file Congruence.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Grid::add_congruence(), Parma_Polyhedra_Library::Box< ITV >::add_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Grid::add_congruence_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_congruence_no_check(), coefficient(), Parma_Polyhedra_Library::Constraint::Constraint(), Parma_Polyhedra_Library::extract_interval_congruence(), is_inconsistent(), is_tautological(), operator<<(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruence(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::shrink_to_congruence_no_check(), Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.25.3.28 void Parma_Polyhedra_Library::Congruence::strong_normalize () [protected]

Calls normalize, then divides out common factors.

Strongly normalized Congruences have equivalent semantics if and only if their syntaxes (as output by operator<<) are equal.

Definition at line 105 of file Congruence.cc.

References normalize().

Referenced by Parma_Polyhedra_Library::Congruence_System::operator<<().

11.25.3.29 void Parma_Polyhedra_Library::Congruence::swap (Congruence & *y*) [inline, private]

Swaps `*this` with `y`.

Definition at line 222 of file Congruence.inlines.hh.

Referenced by Congruence(), and swap().

11.25.3.30 void Parma_Polyhedra_Library::Congruence::throw_dimension_incompatible (const char * *method*, const char * *v_name*, Variable *v*) const [private]

Throws a `std::invalid_argument` exception containing the appropriate error message.

Definition at line 138 of file Congruence.cc.

References Parma_Polyhedra_Library::Variable::space_dimension(), space_dimension(), and throw_invalid_argument().

Referenced by coefficient().

11.25.3.31 void Parma_Polyhedra_Library::Congruence::throw_invalid_argument (const char * *method*, const char * *message*) const [private]

Throws a `std::invalid_argument` exception containing error message `message`.

Definition at line 129 of file Congruence.cc.

Referenced by throw_dimension_incompatible().

11.25.3.32 memory_size_type Parma_Polyhedra_Library::Congruence::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 217 of file Congruence.inlines.hh.

11.25.3.33 const Congruence & Parma_Polyhedra_Library::Congruence::zero_dim_false () [inline, static]

Returns a reference to the false (zero-dimension space) congruence $0 = 1 \pmod 0$.

Definition at line 112 of file Congruence.inlines.hh.

References zero_dim_false_p.

Referenced by Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Congruence_System::initialize(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid::set_empty().

11.25.3.34 const Congruence & Parma_Polyhedra_Library::Congruence::zero_dim_integrality () [inline, static]

Returns a reference to the true (zero-dimension space) congruence $0 = 1 \pmod 1$, also known as the *integrality congruence*.

Definition at line 107 of file Congruence.inlines.hh.

References zero_dim_integrality_p.

Referenced by Parma_Polyhedra_Library::Grid::construct().

11.25.4 Friends And Related Function Documentation

11.25.4.1 bool operator!= (const Congruence & *x*, const Congruence & *y*) [friend]

Returns `false` if and only if `x` and `y` are equivalent.

Definition at line 150 of file Congruence.inlines.hh.

11.25.4.2 Congruence operator%= (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [related]

Returns the congruence $e = n \pmod 1$.

Definition at line 95 of file Congruence.inlines.hh.

References create().

11.25.4.3 Congruence operator%= (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [related]

Returns the congruence $e1 = e2 \pmod 1$.

Definition at line 89 of file Congruence.inlines.hh.

References create().

11.25.4.4 Congruence operator/ (const Constraint & *c*, Coefficient_traits::const_reference *m*) [friend]

Creates a congruence from `c`, with `m` as the modulus.

Definition at line 124 of file Congruence.inlines.hh.

11.25.4.5 Congruence operator/ (const Congruence & *cg*, Coefficient_traits::const_reference *k*) [friend]

Returns a copy of `cg`, multiplying `k` into the copy's modulus.

If `cg` represents the congruence $e_1 = e_2 \pmod{m}$, then the result represents the congruence $e_1 = e_2 \pmod{mk}$.

Definition at line 101 of file Congruence.inlines.hh.

11.25.4.6 std::ostream & operator<< (std::ostream & *s*, const Congruence & *c*) [related]

Output operators.

Definition at line 150 of file Congruence.cc.

References coefficient(), Parma_Polyhedra_Library::Coefficient_zero(), inhomogeneous_term(), is_proper_congruence(), modulus(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and space_dimension().

11.25.4.7 bool operator== (const Congruence & *x*, const Congruence & *y*) [friend]

Returns `true` if and only if `x` and `y` are equivalent.

Definition at line 140 of file Congruence.inlines.hh.

11.25.4.8 friend class Parma_Polyhedra_Library::Congruence_System [friend]

Definition at line 466 of file Congruence.defs.hh.

11.25.4.9 friend class Parma_Polyhedra_Library::Congruence_System::const_iterator [friend]

Definition at line 467 of file Congruence.defs.hh.

11.25.4.10 friend class Parma_Polyhedra_Library::Constraint [friend]

Definition at line 465 of file Congruence.defs.hh.

11.25.4.11 friend class Parma_Polyhedra_Library::Grid [friend]

Definition at line 470 of file Congruence.defs.hh.

11.25.4.12 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Congruence_System & *cgs*) [friend]

11.25.4.13 friend class Parma_Polyhedra_Library::Linear_Expression [friend]

Definition at line 471 of file Congruence.defs.hh.

11.25.4.14 friend class Parma_Polyhedra_Library::Scalar_Products [friend]

Definition at line 464 of file Congruence.defs.hh.

11.25.4.15 void std::swap (Parma_Polyhedra_Library::Congruence & *x*, Parma_Polyhedra_Library::Congruence & *y*) [friend]

11.25.4.16 void swap (Parma_Polyhedra_Library::Congruence & *x*, Parma_Polyhedra_Library::Congruence & *y*) [related]

Specializes `std::swap`.

Definition at line 232 of file Congruence.inlines.hh.

References swap().

11.25.5 Member Data Documentation

11.25.5.1 const PPL::Congruence * Parma_Polyhedra_Library::Congruence::zero_dim_false_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the false (zero-dimension space) congruence $0 = 1 \pmod 0$.

Definition at line 379 of file Congruence.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_false().

11.25.5.2 const PPL::Congruence * Parma_Polyhedra_Library::Congruence::zero_dim_integrality_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the true (zero-dimension space) congruence $0 = 1 \pmod 1$, also known as the *integrality congruence*.

Definition at line 386 of file Congruence.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_integrality().

The documentation for this class was generated from the following files:

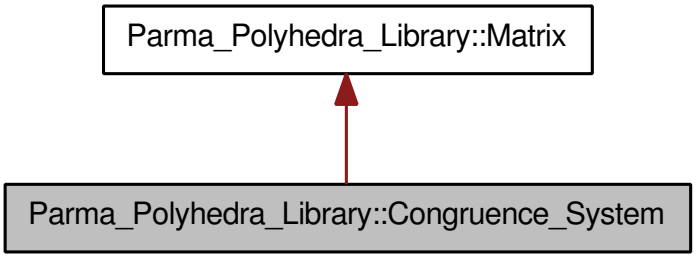
- Congruence.defs.hh
- Congruence.inlines.hh
- Congruence.cc

11.26 Parma_Polyhedra_Library::Congruence_System Class Reference

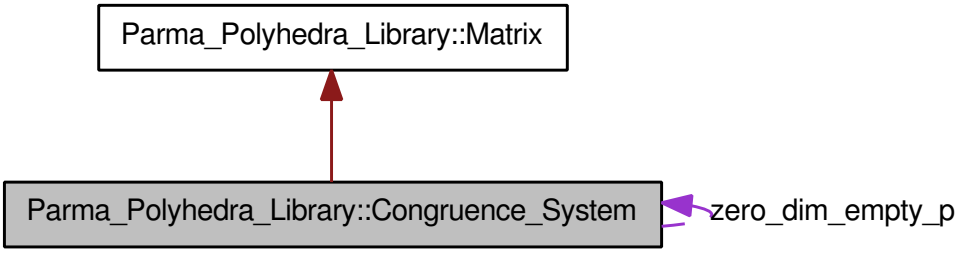
A system of congruences.

`#include <Congruence_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Congruence_System:



Collaboration diagram for Parma_Polyhedra_Library::Congruence_System:



Classes

- class const_iterator

  An iterator over a system of congruences.

Public Member Functions

- Congruence_System ()

  Default constructor: builds an empty system of congruences.

- Congruence_System (const Congruence &cg)

  Builds the singleton system containing only congruence `cg`*.*

- Congruence_System (const Constraint &c)

  If `c` *represents the constraint* $e_1 = e_2$*, builds the singleton system containing only constraint* $e_1 = e_2 \pmod 0$.

- Congruence_System (const Constraint_System &cs)

  Builds a system containing copies of any equalities in `cs`*.*

- Congruence_System (const Congruence_System &cgs)

  Ordinary copy constructor.

- ~Congruence_System ()

  Destructor.

- Congruence_System & operator= (const Congruence_System &cgs)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`*.*

- bool is_equal_to (const Congruence_System &cgs) const

  Returns `true` *if and only if* `*this` *is exactly equal to* `cgs`*.*

- bool has_linear_equalities () const

  Returns `true` *if and only if* `*this` *contains one or more linear equalities.*

- void clear ()

  Removes all the congruences and sets the space dimension to 0.

- void insert (const Congruence &cg)

  Inserts in `*this` *a copy of the congruence* `cg`*, increasing the number of space dimensions if needed.*

- void insert (const Constraint &c)

  Inserts in `*this` *a copy of the equality constraint* `c`*, seen as a modulo 0 congruence, increasing the number of space dimensions if needed.*

- void insert (const Congruence_System &cgs)

Inserts in `*this` *a copy of the congruences in* `cgs`*, increasing the number of space dimensions if needed.*

- void recycling_insert (Congruence_System &cgs)

  Inserts into `*this` *the congruences in* `cgs`*, increasing the number of space dimensions if needed.*

- bool empty () const

  Returns `true` *if and only if* `*this` *has no congruences.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first congruence, if `this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`*.*

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`*.*

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`*.*

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`*.*

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`*.*

- dimension_type num_equalities () const

  Returns the number of equalities.

- dimension_type num_proper_congruences () const

  Returns the number of proper congruences.

- void swap (Congruence_System &cgs)

  Swaps `*this` *with* `y`*.*

- void add_unit_rows_and_columns (dimension_type dims)

  Adds `dims` *rows and* `dims` *columns of zeroes to the matrix, initializing the added rows as in the unit congruence system.*

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Congruence_System can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Congruence_System & zero_dim_empty ()

  Returns the system containing only Congruence::zero_dim_false().

Protected Member Functions

- bool satisfies_all_congruences (const Grid_Generator &g) const

  Returns `true` *if* `g` *satisfies all the congruences.*

Private Member Functions

- Congruence_System (dimension_type d)

  Builds an empty (i.e. zero rows) system of dimension `d`.

- void concatenate (const Congruence_System &cgs)

  Concatenates copies of the congruences from `cgs` *onto* `*this`.

- void normalize_moduli ()

  Adjusts all expressions to have the same moduli.

- bool increase_space_dimension (dimension_type new_space_dim)

  Increase the number of space dimensions to `new_space_dim`.

- void insert_verbatim (const Congruence &cg)

  Inserts in `*this` *an exact copy of the congruence* `cg`*, increasing the number of space dimensions if needed.*

- Congruence & operator[ ] (dimension_type k)

  Returns the `k`*− th congruence of the system.*

- const Congruence & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k`*− th congruence of the system.*

- bool has_a_free_dimension () const

  Returns `true` *if and only if any of the dimensions in* `*this` *is free of constraint.*

- void affine_preimage (dimension_type v, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator)

  Substitutes a given column of coefficients by a given affine expression.

- void remove_higher_space_dimensions (dimension_type new_dimension)

  Removes the higher dimensions of the system so that the resulting system will have dimension `new_dimension`.

- void resize_no_copy (dimension_type new_num_rows, dimension_type new_num_columns)

  Resizes the system without worrying about the old contents.

Static Private Attributes

- static const Congruence_System * zero_dim_empty_p = 0

  Holds (between class initialization and finalization) a pointer to the singleton system containing only Congruence::zero_dim_false().

Friends

- class const_iterator
- class Grid
- class Grid_Certificate
- void std::swap (Parma_Polyhedra_Library::Congruence_System &x, Parma_Polyhedra_Library::Congruence_System &y)
- bool operator== (const Congruence_System &x, const Congruence_System &y)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Congruence_System &cgs)

  Output operator.

- void swap (Parma_Polyhedra_Library::Congruence_System &x, Parma_Polyhedra_Library::Congruence_System &y)

  Specializes `std::swap`.

11.26.1 Detailed Description

A system of congruences. An object of the class Congruence_System is a system of congruences, i.e., a multiset of objects of the class Congruence. When inserting congruences in a system, space dimensions are automatically adjusted so that all the congruences in the system are defined on the same vector space.

In all the examples it is assumed that variables `x` and `y` are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a system of congruences corresponding to an integer grid in $\mathbb{R}^2$:

```
Congruence_System cgs;
cgs.insert(x %= 0);
cgs.insert(y %= 0);
```

Note that: the congruence system is created with space dimension zero; the first and second congruence insertions increase the space dimension to 1 and 2, respectively.

Example 2

By adding to the congruence system of the previous example, the congruence $x + y = 1 \pmod 2$:

```
cgs.insert((x + y %= 1) / 2);
```

we obtain the grid containing just those integral points where the sum of the x and y values is odd.

Example 3

The following code builds a system of congruences corresponding to the grid in $\mathbb{Z}^2$ containing just the integral points on the x axis:

```
Congruence_System cgs;
cgs.insert(x %= 0);
cgs.insert((y %= 0) / 0);
```

Note

After inserting a multiset of congruences in a congruence system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* congruence system will be available, where original congruences may have been reordered, removed (if they are trivial, duplicate or implied by other congruences), linearly combined, etc.

Definition at line 122 of file Congruence_System.defs.hh.

11.26.2 Constructor & Destructor Documentation

11.26.2.1 Parma_Polyhedra_Library::Congruence_System::Congruence_System () [inline]

Default constructor: builds an empty system of congruences.

Definition at line 49 of file Congruence_System.inlines.hh.

Referenced by initialize().

11.26.2.2 Parma_Polyhedra_Library::Congruence_System::Congruence_System (const Congruence & *cg*) [inline, explicit]

Builds the singleton system containing only congruence cg.

Definition at line 54 of file Congruence_System.inlines.hh.

References insert().

11.26.2.3 Parma_Polyhedra_Library::Congruence_System::Congruence_System (const Constraint & *c*) [inline, explicit]

If c represents the constraint $e_1 = e_2$, builds the singleton system containing only constraint $e_1 = e_2 \pmod 0$.

Exceptions

std::invalid_argument Thrown if c is not an equality constraint.

Definition at line 60 of file Congruence_System.inlines.hh.

References insert().

11.26.2.4 Parma_Polyhedra_Library::Congruence_System::Congruence_System (const Constraint_System & *cs*) [explicit]

Builds a system containing copies of any equalities in cs.

Definition at line 41 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), and insert().

11.26.2.5 Parma_Polyhedra_Library::Congruence_System::Congruence_System (const Congruence_System & *cgs*) [inline]

Ordinary copy constructor.

Definition at line 66 of file Congruence_System.inlines.hh.

11.26.2.6 Parma_Polyhedra_Library::Congruence_System::~Congruence_System () [inline]

Destructor.

Definition at line 76 of file Congruence_System.inlines.hh.

11.26.2.7 Parma_Polyhedra_Library::Congruence_System::Congruence_System (dimension_type *d*) [inline, explicit, private]

Builds an empty (i.e. zero rows) system of dimension d.

Definition at line 71 of file Congruence_System.inlines.hh.

11.26.3 Member Function Documentation

11.26.3.1 void Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns (dimension_type *dims*)

Adds `dims` rows and `dims` columns of zeroes to the matrix, initializing the added rows as in the unit congruence system.

Parameters

dims The number of rows and columns to be added: must be strictly positive.

Turns the $r \times c$ matrix A into the $(r + dims) \times (c + dims)$ matrix $\left(\begin{smallmatrix} 0 & B \\ A & A \end{smallmatrix}\right)$ where B is the $dims \times dims$ unit matrix of the form $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 507 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Linear_Row::LINE_OR_EQUALITY, Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::swap(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::add_space_dimensions(), and Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project().

11.26.3.2 void Parma_Polyhedra_Library::Congruence_System::affine_preimage (dimension_type *v*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator*) [private]

Substitutes a given column of coefficients by a given affine expression.

Parameters

v Index of the column to which the affine transformation is substituted;

expr The numerator of the affine transformation: $\sum_{i=0}^{n-1} a_i x_i + b$;

denominator The denominator of the affine transformation.

We allow affine transformations (see the Section Operations on Rational Grids) to have rational coefficients. Since the coefficients of linear expressions are integers we also provide an integer `denominator` that will be used as denominator of the affine transformation. The denominator is required to be a positive integer and its default value is 1.

The affine transformation substitutes the matrix of congruences by a new matrix whose elements a'_{ij} are built from the old one a_{ij} as follows:

$$a'_{ij} = \begin{cases} a_{ij} * \text{denominator} + a_{iv} * \text{expr}[j] & \text{for } j \neq v; \\ \text{expr}[v] * a_{iv} & \text{for } j = v. \end{cases}$$

`expr` is a constant parameter and unaltered by this computation.

Definition at line 339 of file Congruence_System.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and space_dimension().

Referenced by Parma_Polyhedra_Library::Grid::affine_image(), and Parma_Polyhedra_Library::Grid::affine_preimage().

11.26.3.3 void Parma_Polyhedra_Library::Congruence_System::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 392 of file Congruence_System.cc.

References ascii_dump(), Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

11.26.3.4 void Parma_Polyhedra_Library::Congruence_System::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Referenced by Parma_Polyhedra_Library::Grid::ascii_dump(), ascii_dump(), and Parma_Polyhedra_Library::Grid::OK().

11.26.3.5 bool Parma_Polyhedra_Library::Congruence_System::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 406 of file Congruence_System.cc.

References Parma_Polyhedra_Library::ascii_load(), and Parma_Polyhedra_Library::Matrix::num_rows().

11.26.3.6 Congruence_System::const_iterator Parma_Polyhedra_Library::Congruence_System::begin () const [inline]

Returns the const_iterator pointing to the first congruence, if `this` is not empty; otherwise, returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 176 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::begin().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Grid::add_recycled_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Constraint_System::Constraint_System(), Parma_Polyhedra_Library::Grid::difference_assign(), empty(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_-

congruences(), operator<<(), Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), and Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences().

11.26.3.7 void Parma_Polyhedra_Library::Congruence_System::clear () [inline]

Removes all the congruences and sets the space dimension to 0.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 96 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::add_zero_columns().

Referenced by Parma_Polyhedra_Library::Grid::set_zero_dim_univ().

11.26.3.8 void Parma_Polyhedra_Library::Congruence_System::concatenate (const Congruence_System & *cgs*) [private]

Concatenates copies of the congruences from `cgs` onto `*this`.

Parameters

cgs The congruence system to append to `this`. The number of rows in `cgs` must be strictly positive.

The matrix for the new system of congruences is obtained by leaving the old system in the upper left-hand side and placing the congruences of `cgs` in the lower right-hand side, and padding with zeroes.

Definition at line 528 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), operator[ ](), space_dimension(), Parma_Polyhedra_Library::swap(), and swap().

Referenced by Parma_Polyhedra_Library::Grid::concatenate_assign().

11.26.3.9 bool Parma_Polyhedra_Library::Congruence_System::empty () const [inline]

Returns `true` if and only if `*this` has no congruences.

Definition at line 189 of file Congruence_System.inlines.hh.

References begin(), and end().

11.26.3.10 Congruence_System::const_iterator Parma_Polyhedra_Library::Congruence_System::end () const [inline]

Returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 183 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::end().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Grid::add_recycled_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Constraint_System::Constraint_System(), Parma_Polyhedra_Library::Grid::difference_assign(), empty(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), operator<<(), Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), and Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences().

11.26.3.11 memory_size_type Parma_Polyhedra_Library::Congruence_System::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 199 of file Congruence_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::external_memory_in_bytes().

11.26.3.12 void Parma_Polyhedra_Library::Congruence_System::finalize () [static]

Finalizes the class.

Definition at line 438 of file Congruence_System.cc.

References zero_dim_empty_p.

11.26.3.13 bool Parma_Polyhedra_Library::Congruence_System::has_a_free_dimension () const [private]

Returns `true` if and only if any of the dimensions in `*this` is free of constraint.

Any equality or proper congruence affecting a dimension constrains that dimension.

This method assumes the system is in minimal form.

Definition at line 307 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), operator[ ](), and space_dimension().

11.26.3.14 bool Parma_Polyhedra_Library::Congruence_System::has_linear_equalities () const

Returns `true` if and only if `*this` contains one or more linear equalities.

Definition at line 238 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

11.26.3.15 bool Parma_Polyhedra_Library::Congruence_System::increase_space_dimension (dimension_type *new_space_dim*) [private]

Increase the number of space dimensions to `new_space_dim`.

`new_space_dim` must at least equal to the current space dimension.

Definition at line 51 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), space_dimension(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid::set_empty().

11.26.3.16 void Parma_Polyhedra_Library::Congruence_System::initialize () [static]

Initializes the class.

Definition at line 431 of file Congruence_System.cc.

References Congruence_System(), zero_dim_empty_p, and Parma_Polyhedra_Library::Congruence::zero_dim_false().

11.26.3.17 void Parma_Polyhedra_Library::Congruence_System::insert (const Congruence_System & *cgs*)

Inserts in `*this` a copy of the congruences in `cgs`, increasing the number of space dimensions if needed.

The inserted copies will be strongly normalized.

Definition at line 156 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, swap(), and Parma_Polyhedra_Library::Matrix::swap_columns().

11.26.3.18 void Parma_Polyhedra_Library::Congruence_System::insert (const Constraint & *c*)

Inserts in `*this` a copy of the equality constraint `c`, seen as a modulo 0 congruence, increasing the number of space dimensions if needed.

The modulo 0 congruence will be strongly normalized after being inserted.

Exceptions

std::invalid_argument Thrown if `c` is a relational constraint.

Definition at line 100 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_recycled_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), operator[ ](), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::rows, Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Matrix::swap_columns().

11.26.3.19 void Parma_Polyhedra_Library::Congruence_System::insert (const Congruence & *cg*) [inline]

Inserts in `*this` a copy of the congruence `cg`, increasing the number of space dimensions if needed.

The copy of `cg` will be strongly normalized after being inserted.

Definition at line 42 of file Congruence_System.inlines.hh.

References insert_verbatim(), OK(), operator[ ](), and Parma_Polyhedra_Library::Matrix::rows.

Referenced by Parma_Polyhedra_Library::Grid::add_congruence_no_check(), Congruence_System(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::contains_integer_point(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Grid::intersection_assign(), Parma_Polyhedra_Library::Grid::limited_congruence_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_generator_extrapolation_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Grid::select_wider_congruences(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.26.3.20 void Parma_Polyhedra_Library::Congruence_System::insert_verbatim (const Congruence & *cg*) [private]

Inserts in `*this` an exact copy of the congruence `cg`, increasing the number of space dimensions if needed.

This method inserts a copy of `cg` in the given form, instead of first strong normalizing `cg` as insert would do.

Definition at line 73 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_recycled_row(), Parma_Polyhedra_Library::Matrix::add_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Row::size(), swap(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::expand_space_dimension(), and insert().

11.26.3.21 bool Parma_Polyhedra_Library::Congruence_System::is_equal_to (const Congruence_System & *cgs*) const

Returns `true` if and only if `*this` is exactly equal to `cgs`.

Definition at line 224 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Grid::OK().

11.26.3.22 dimension_type Parma_Polyhedra_Library::Congruence_System::max_space_dimension () [inline, static]

Returns the maximum space dimension a Congruence_System can handle.

Definition at line 86 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::max_num_columns().

Referenced by Parma_Polyhedra_Library::Grid::max_space_dimension().

11.26.3.23 void Parma_Polyhedra_Library::Congruence_System::normalize_moduli () [private]

Adjusts all expressions to have the same moduli.

Definition at line 187 of file Congruence_System.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::lcm_assign(), Parma_Polyhedra_Library::Congruence::modulus(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), operator[ ](), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Matrix::row_size, and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Grid::construct(), and Parma_Polyhedra_Library::Grid::simplify().

11.26.3.24 PPL::dimension_type Parma_Polyhedra_Library::Congruence_System::num_equalities () const

Returns the number of equalities.

Definition at line 255 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Grid::congruence_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), and Parma_Polyhedra_Library::Grid::quick_equivalence_test().

11.26.3.25 PPL::dimension_type Parma_Polyhedra_Library::Congruence_System::num_proper_congruences () const

Returns the number of proper congruences.

Definition at line 265 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Congruence::is_proper_congruence(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), and Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate().

11.26.3.26 bool Parma_Polyhedra_Library::Congruence_System::OK () const

Checks if all the invariants are satisfied.

Returns `true` if and only if `*this` is a valid Matrix, each row in the system is a valid Congruence and the number of columns is consistent with the number of congruences.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 445 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Congruence::OK(), and Parma_Polyhedra_Library::Matrix::OK().

Referenced by increase_space_dimension(), insert(), insert_verbatim(), normalize_moduli(), Parma_Polyhedra_Library::Grid::OK(), recycling_insert(), remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid::simplify().

11.26.3.27 Congruence_System & Parma_Polyhedra_Library::Congruence_System::operator= (const Congruence_System & *cgs*) `[inline]`

Assignment operator.

Definition at line 80 of file Congruence_System.inlines.hh.

11.26.3.28 const Congruence & Parma_Polyhedra_Library::Congruence_System::operator[ ] (dimension_type *k*) const `[inline, private]`

Returns a constant reference to the `k`− th congruence of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 37 of file Congruence_System.inlines.hh.

References operator[ ]().

11.26.3.29 Congruence & Parma_Polyhedra_Library::Congruence_System::operator[ ] (dimension_type *k*) [inline, private]

Returns the `k`− th congruence of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 32 of file Congruence_System.inlines.hh.

Referenced by concatenate(), has_a_free_dimension(), insert(), normalize_moduli(), operator[ ](), and recycling_insert().

11.26.3.30 void Parma_Polyhedra_Library::Congruence_System::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

11.26.3.31 void Parma_Polyhedra_Library::Congruence_System::recycling_insert (Congruence_System & *cgs*)

Inserts into `*this` the congruences in `cgs`, increasing the number of space dimensions if needed.

Definition at line 125 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), operator[ ](), swap(), Parma_Polyhedra_Library::swap(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::add_recycled_congruences().

11.26.3.32 void Parma_Polyhedra_Library::Congruence_System::remove_higher_space_dimensions (dimension_type *new_dimension*) [private]

Removes the higher dimensions of the system so that the resulting system will have dimension `new_dimension`.

The value of `new_dimension` must be at most the space dimension of `*this`.

Definition at line 566 of file Congruence_System.cc.

References OK(), Parma_Polyhedra_Library::Matrix::remove_trailing_columns(), space_dimension(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions().

11.26.3.33 void Parma_Polyhedra_Library::Congruence_System::resize_no_copy (dimension_type *new_num_rows*, dimension_type *new_num_columns*) [inline, private]

Resizes the system without worrying about the old contents.

Parameters

new_num_rows The number of rows of the resized system;

new_num_columns The number of columns of the resized system.

The system is expanded to the specified dimensions avoiding reallocation whenever possible. The contents of the original system is lost.

Definition at line 102 of file Congruence_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::conversion().

11.26.3.34 bool Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences (const Grid_Generator & *g*) const [protected]

Returns `true` if `g` satisfies all the congruences.

Definition at line 278 of file Congruence_System.cc.

References Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Grid_Generator::is_line(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, space_dimension(), and Parma_Polyhedra_Library::Grid_Generator::space_dimension().

Referenced by Parma_Polyhedra_Library::Grid::is_included_in(), Parma_Polyhedra_Library::Grid::is_universe(), and Parma_Polyhedra_Library::Grid::relation_with().

11.26.3.35 dimension_type Parma_Polyhedra_Library::Congruence_System::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 91 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::num_columns().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruences(), Parma_Polyhedra_Library::Grid::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_congruences(), Parma_Polyhedra_Library::Box< ITV >::add_congruences_no_check(), Parma_Polyhedra_Library::Grid::add_recycled_congruences(), affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::Box(), concatenate(), Parma_Polyhedra_Library::Grid::construct(), has_a_free_dimension(), increase_space_dimension(), Parma_Polyhedra_Library::Grid::limited_congruence_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_generator_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Grid::OK(), Parma_Polyhedra_Library::Box< ITV >::refine_-

no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::Box< ITV >::refine_with_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_congruences(), remove_higher_space_dimensions(), satisfies_all_congruences(), Parma_Polyhedra_Library::Grid::select_wider_congruences(), Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible().

11.26.3.36 void Parma_Polyhedra_Library::Congruence_System::swap (Congruence_System & *cgs*) [inline]

Swaps `*this` with `y`.

Definition at line 194 of file Congruence_System.inlines.hh.

Referenced by concatenate(), Parma_Polyhedra_Library::Grid::construct(), insert(), insert_verbatim(), recycling_insert(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), and swap().

11.26.3.37 memory_size_type Parma_Polyhedra_Library::Congruence_System::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 204 of file Congruence_System.inlines.hh.

11.26.3.38 const Congruence_System & Parma_Polyhedra_Library::Congruence_System::zero_dim_empty () [inline, static]

Returns the system containing only Congruence::zero_dim_false().

Definition at line 108 of file Congruence_System.inlines.hh.

References zero_dim_empty_p.

Referenced by Parma_Polyhedra_Library::Box< ITV >::congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), and Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences().

11.26.4 Friends And Related Function Documentation

11.26.4.1 friend class const_iterator [friend]

Definition at line 407 of file Congruence_System.defs.hh.

11.26.4.2 friend class Grid [friend]

Definition at line 408 of file Congruence_System.defs.hh.

11.26.4.3 friend class Grid_Certificate [friend]

Definition at line 409 of file Congruence_System.defs.hh.

11.26.4.4 std::ostream & operator<< (std::ostream & *s*, const Congruence_System & *cgs*) [related]

Output operator.

Writes `true` if `cgs` is empty. Otherwise, writes on `s` the congruences of `cgs`, all in one row and separated by ", ".

Definition at line 474 of file Congruence_System.cc.

References begin(), end(), and Parma_Polyhedra_Library::Congruence::strong_normalize().

11.26.4.5 bool operator== (const Congruence_System & *x*, const Congruence_System & *y*) [friend]

Definition at line 491 of file Congruence_System.cc.

11.26.4.6 void std::swap (Parma_Polyhedra_Library::Congruence_System & *x*, Parma_Polyhedra_Library::Congruence_System & *y*) [friend]

11.26.4.7 void swap (Parma_Polyhedra_Library::Congruence_System & *x*, Parma_Polyhedra_Library::Congruence_System & *y*) [related]

Specializes `std::swap`.

Definition at line 215 of file Congruence_System.inlines.hh.

References swap().

11.26.5 Member Data Documentation

11.26.5.1 const PPL::Congruence_System * Parma_Polyhedra_Library::Congruence_System::zero_dim_empty_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the singleton system containing only Congruence::zero_dim_false().

Definition at line 369 of file Congruence_System.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_empty().

The documentation for this class was generated from the following files:

- Congruence_System.defs.hh
- Congruence.defs.hh
- Congruence_System.inlines.hh
- Congruence_System.cc

11.27 Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 > Class Template Reference

This class provides the reduction method for the Congruences_Product domain.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- Congruences_Reduction ()
  Default constructor.
- void product_reduce (D1 &d1, D2 &d2)
  The congruences reduction operator for detect emptiness or any equalities implied by each of the congruences defining one of the components and the bounds of the other component. It is assumed that the components are already constraints reduced.
- ~Congruences_Reduction ()
  Destructor.

11.27.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >

This class provides the reduction method for the Congruences_Product domain. The reduction classes are used to instantiate the Partially_Reduced_Product domain.

This class uses the minimized congruences defining each of the components. For each of the congruences, it checks if the other component intersects none, one or more than one hyperplane defined by the congruence and adds equalities or emptiness as appropriate; in more detail: Letting the components be d1 and d2, then, for each congruence cg representing d1:

- if more than one hyperplane defined by cg intersects d2, then d1 and d2 are unchanged;
- if exactly one hyperplane intersects d2, then d1 and d2 are refined with the corresponding equality ;
- otherwise, d1 and d2 are set to empty. Unless d1 and d2 are already empty, the process is repeated where the roles of d1 and d2 are reversed. If d1 or d2 is empty, then the emptiness is propagated.

Definition at line 193 of file Partially_Reduced_Product.defs.hh.

11.27.2 Constructor & Destructor Documentation

11.27.2.1 template<typename D1 , typename D2 > Parma_Polyhedra_-Library::Congruences_Reduction< D1, D2 >::Congruences_Reduction () [inline]

Default constructor.

Definition at line 781 of file Partially_Reduced_Product.inlines.hh.

11.27.2.2 template<typename D1 , typename D2 > Parma_Polyhedra_-Library::Congruences_Reduction< D1, D2 >::~Congruences_Reduction () [inline]

Destructor.

Definition at line 786 of file Partially_Reduced_Product.inlines.hh.

11.27.3 Member Function Documentation

11.27.3.1 template<typename D1 , typename D2 > void Parma_Polyhedra_-Library::Congruences_Reduction< D1, D2 >::product_reduce (D1 & *d1*, D2 & *d2*) [inline]

The congruences reduction operator for detect emptiness or any equalities implied by each of the congruences defining one of the components and the bounds of the other component. It is assumed that the components are already constraints reduced.

The minimized congruence system defining the domain element `d1` is used to check if `d2` intersects none, one or more than one of the hyperplanes defined by the congruences: if it intersects none, then product is set empty; if it intersects one, then the equality defining this hyperplane is added to both components; otherwise, the product is unchanged. In each case, the donor domain must provide a congruence system in minimal form.

Parameters

- ***d1*** A pointset domain element;
- ***d2*** A pointset domain element;

Definition at line 579 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_-Library::Congruence_System::end(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_-Polyhedra_Library::Smash_Reduction< D1, D2 >::product_reduce(), and Parma_Polyhedra_-Library::shrink_to_congruence_no_check().

Referenced by Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce().

The documentation for this class was generated from the following files:

- Partially_Reduced_Product.defs.hh

- Partially_Reduced_Product.inlines.hh
- Partially_Reduced_Product.templates.hh

11.28 Parma_Polyhedra_Library::Matrix::const_iterator Class Reference

An iterator over a matrix.

`#include <Matrix.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Matrix::const_iterator:

Public Types

- typedef std::forward_iterator_tag iterator_category
- typedef std::iterator_traits< Iter >::value_type value_type
- typedef std::iterator_traits< Iter >::difference_type difference_type
- typedef std::iterator_traits< Iter >::pointer pointer
- typedef std::iterator_traits< Iter >::reference reference

Public Member Functions

- const_iterator ()

  Default constructor.

- const_iterator (const Iter &b)

  Builds a const iterator on the matrix starting from an iterator `b` *on the elements of the vector* `rows`*.*

- const_iterator (const const_iterator &y)

  Ordinary copy constructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- reference operator* () const

  Dereference operator.

- pointer operator-> () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Types

- typedef std::vector< Row >::const_iterator Iter

Private Attributes

- Iter i

  The const iterator on the rows' vector `rows`*.*

11.28.1 Detailed Description

An iterator over a matrix. A const_iterator is used to provide read-only access to each row contained in a Matrix object.

Definition at line 91 of file Matrix.defs.hh.

11.28.2 Member Typedef Documentation

11.28.2.1 typedef std::iterator_traits<Iter>::difference_type Parma_Polyhedra_Library::Matrix::const_iterator::difference_type

Definition at line 100 of file Matrix.defs.hh.

11.28.2.2 typedef std::vector<Row>::const_iterator Parma_Polyhedra_Library::Matrix::const_iterator::Iter [private]

Definition at line 93 of file Matrix.defs.hh.

11.28.2.3 typedef std::forward_iterator_tag Parma_Polyhedra_Library::Matrix::const_iterator::iterator_category

Definition at line 98 of file Matrix.defs.hh.

11.28.2.4 typedef std::iterator_traits<Iter>::pointer Parma_Polyhedra_Library::Matrix::const_iterator::pointer

Definition at line 101 of file Matrix.defs.hh.

11.28.2.5 typedef std::iterator_traits<Iter>::reference Parma_Polyhedra_Library::Matrix::const_iterator::reference

Definition at line 102 of file Matrix.defs.hh.

11.28.2.6 typedef std::iterator_traits<Iter>::value_type Parma_Polyhedra_Library::Matrix::const_iterator::value_type

Definition at line 99 of file Matrix.defs.hh.

11.28.3 Constructor & Destructor Documentation

11.28.3.1 Parma_Polyhedra_Library::Matrix::const_iterator::const_iterator () [inline]

Default constructor.

Definition at line 48 of file Matrix.inlines.hh.

Referenced by operator++().

11.28.3.2 Parma_Polyhedra_Library::Matrix::const_iterator::const_iterator (const Iter & *b*) [inline, explicit]

Builds a const iterator on the matrix starting from an iterator `b` on the elements of the vector `rows`.

Definition at line 53 of file Matrix.inlines.hh.

11.28.3.3 Parma_Polyhedra_Library::Matrix::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Definition at line 58 of file Matrix.inlines.hh.

11.28.4 Member Function Documentation

11.28.4.1 bool Parma_Polyhedra_Library::Matrix::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 95 of file Matrix.inlines.hh.

References operator==().

11.28.4.2 Matrix::const_iterator::reference Parma_Polyhedra_Library::Matrix::const_iterator::operator∗ () const [inline]

Dereference operator.

Definition at line 69 of file Matrix.inlines.hh.

References i.

11.28.4.3 Matrix::const_iterator Parma_Polyhedra_Library::Matrix::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Definition at line 85 of file Matrix.inlines.hh.

References const_iterator(), and i.

11.28.4.4 Matrix::const_iterator & Parma_Polyhedra_Library::Matrix::const_iterator::operator++ () [inline]

Prefix increment operator.

Definition at line 79 of file Matrix.inlines.hh.

References i.

11.28.4.5 Matrix::const_iterator::pointer Parma_Polyhedra_Library::Matrix::const_iterator::operator-> () const [inline]

Indirect member selector.

Definition at line 74 of file Matrix.inlines.hh.

References i.

11.28.4.6 Matrix::const_iterator & Parma_Polyhedra_Library::Matrix::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Definition at line 63 of file Matrix.inlines.hh.

References i.

11.28.4.7 bool Parma_Polyhedra_Library::Matrix::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 90 of file Matrix.inlines.hh.

References i.

Referenced by operator!=().

11.28.5 Member Data Documentation

11.28.5.1 Iter Parma_Polyhedra_Library::Matrix::const_iterator::i [private]

The const iterator on the rows' vector `rows`.

Definition at line 95 of file Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::Generator_System::const_iterator::operator!=(), Parma_Polyhedra_Library::Constraint_System::const_iterator::operator!=(), Parma_Polyhedra_Library::Congruence_System::const_iterator::operator!=(), operator*(), operator++(), operator->(), operator=(), Parma_Polyhedra_Library::Generator_System::const_iterator::operator=(), Parma_Polyhedra_Library::Constraint_System::const_iterator::operator=(), Parma_Polyhedra_Library::Congruence_System::const_iterator::operator=(), operator==(), Parma_Polyhedra_Library::Generator_System::const_iterator::operator==(), Parma_Polyhedra_Library::Constraint_System::const_iterator::operator==(), and Parma_Polyhedra_Library::Congruence_System::const_iterator::operator==().

The documentation for this class was generated from the following files:

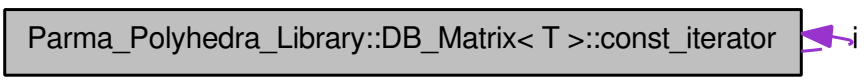
- Matrix.defs.hh
- Matrix.inlines.hh

11.29 Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator Class Reference

A read-only iterator over the rows of the matrix.

`#include <DB_Matrix.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator:



Public Types

- typedef std::forward_iterator_tag iterator_category
- typedef std::iterator_traits< Iter >::value_type value_type
- typedef std::iterator_traits< Iter >::difference_type difference_type

- typedef std::iterator_traits< Iter >::pointer pointer
- typedef std::iterator_traits< Iter >::reference reference

Public Member Functions

- const_iterator ()

  Default constructor.

- const_iterator (const Iter &b)

  Builds a const iterator on the matrix starting from an iterator `b` *on the elements of the vector* `rows`*.*

- const_iterator (const const_iterator &y)

  Ordinary copy constructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- reference operator* () const

  Dereference operator.

- pointer operator-> () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Types

- typedef std::vector< DB_Row< T > >::const_iterator Iter

Private Attributes

- Iter i

  The const iterator on the rows' vector `rows`*.*

11.29.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator

A read-only iterator over the rows of the matrix.

Definition at line 96 of file DB_Matrix.defs.hh.

11.29.2 Member Typedef Documentation

11.29.2.1 template<typename T> typedef std::iterator_traits<Iter>::difference_type Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::difference_type

Definition at line 106 of file DB_Matrix.defs.hh.

11.29.2.2 template<typename T> typedef std::vector<DB_Row<T> >::const_iterator Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::Iter [private]

Definition at line 98 of file DB_Matrix.defs.hh.

11.29.2.3 template<typename T> typedef std::forward_iterator_tag Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::iterator_category

Definition at line 103 of file DB_Matrix.defs.hh.

11.29.2.4 template<typename T> typedef std::iterator_traits<Iter>::pointer Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::pointer

Definition at line 107 of file DB_Matrix.defs.hh.

11.29.2.5 template<typename T> typedef std::iterator_traits<Iter>::reference Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::reference

Definition at line 108 of file DB_Matrix.defs.hh.

11.29.2.6 template<typename T> typedef std::iterator_traits<Iter>::value_type Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::value_type

Definition at line 104 of file DB_Matrix.defs.hh.

11.29.3 Constructor & Destructor Documentation

11.29.3.1 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::const_iterator () [inline]

Default constructor.

Definition at line 63 of file DB_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator++().

11.29.3.2 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::const_iterator (const Iter & *b*) [inline, explicit]

Builds a const iterator on the matrix starting from an iterator `b` on the elements of the vector `rows`.

Definition at line 69 of file DB_Matrix.inlines.hh.

11.29.3.3 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Definition at line 75 of file DB_Matrix.inlines.hh.

11.29.4 Member Function Documentation

11.29.4.1 template<typename T > bool Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 119 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator==().

11.29.4.2 template<typename T > DB_Matrix< T >::const_iterator::reference Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator* () const [inline]

Dereference operator.

Definition at line 88 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

11.29.4.3 template<typename T > DB_Matrix< T >::const_iterator Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Definition at line 107 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::const_iterator(), and Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

11.29.4.4 template<typename T > DB_Matrix< T >::const_iterator & Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator++ () [inline]

Prefix increment operator.

Definition at line 100 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

11.29.4.5 template<typename T > DB_Matrix< T >::const_iterator::pointer Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator-> () const [inline]

Indirect member selector.

Definition at line 94 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

11.29.4.6 template<typename T > DB_Matrix< T >::const_iterator & Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Definition at line 81 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

11.29.4.7 template<typename T > bool Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 113 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator!=().

11.29.5 Member Data Documentation

11.29.5.1 template<typename T> Iter Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::i [private]

The const iterator on the rows' vector `rows`.

Definition at line 100 of file DB_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator*(), Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator++(), Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator->(), Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator=(), and Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator::operator==().

The documentation for this class was generated from the following files:

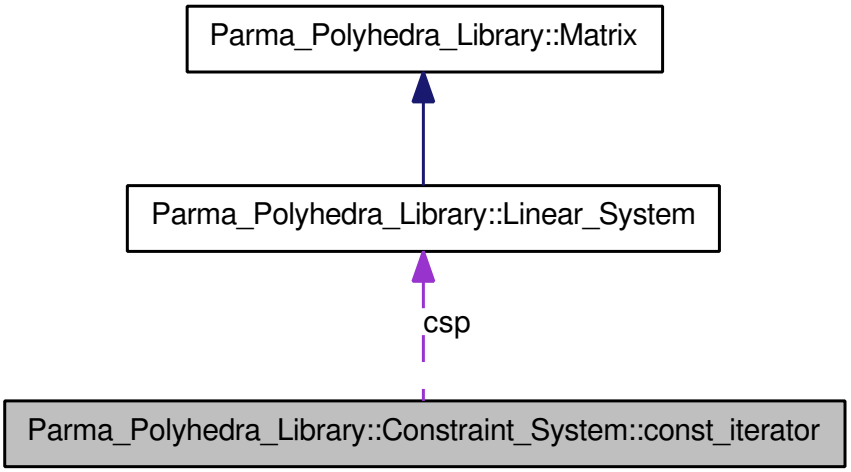
- DB_Matrix.defs.hh
- DB_Matrix.inlines.hh

11.30 Parma_Polyhedra_Library::Constraint_System::const_iterator Class Reference

An iterator over a system of constraints.

`#include <Constraint_System.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Constraint_System::const_iterator:



Public Member Functions

- const_iterator ()

  Default constructor.

- const_iterator (const const_iterator &y)

  Ordinary copy constructor.

- ~const_iterator ()

  Destructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- const Constraint & operator* () const

  Dereference operator.

- const Constraint * operator-> () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` if and only if `*this` and `y` are identical.

- bool operator!= (const const_iterator &y) const

  Returns `true` if and only if `*this` and `y` are different.

Private Member Functions

- const_iterator (const Linear_System::const_iterator &iter, const Constraint_System &csys)

  Constructor.

- void skip_forward ()

  `*this` skips to the next non-trivial constraint.

Private Attributes

- Linear_System::const_iterator i

  The const iterator over the matrix of constraints.

- const Linear_System * csp

  A const pointer to the matrix of constraints.

Friends

- class Constraint_System

11.30.1 Detailed Description

An iterator over a system of constraints. A const_iterator is used to provide read-only access to each constraint contained in a Constraint_System object.

Example

The following code prints the system of constraints defining the polyhedron `ph`:

```
const Constraint_System& cs = ph.constraints();
for (Constraint_System::const_iterator i = cs.begin(),
       cs_end = cs.end(); i != cs_end; ++i)
  cout << *i << endl;
```

Definition at line 204 of file Constraint_System.defs.hh.

11.30.2 Constructor & Destructor Documentation

11.30.2.1 Parma_Polyhedra_Library::Constraint_System::const_iterator::const_iterator () [inline]

Default constructor.

Definition at line 101 of file Constraint_System.inlines.hh.

11.30.2.2 Parma_Polyhedra_Library::Constraint_System::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Definition at line 106 of file Constraint_System.inlines.hh.

11.30.2.3 Parma_Polyhedra_Library::Constraint_System::const_iterator::~const_iterator () [inline]

Destructor.

Definition at line 111 of file Constraint_System.inlines.hh.

11.30.2.4 Parma_Polyhedra_Library::Constraint_System::const_iterator::const_iterator (const Linear_System::const_iterator & *iter*, const Constraint_System & *csys*) [inline, private]

Constructor.

Definition at line 157 of file Constraint_System.inlines.hh.

11.30.3 Member Function Documentation

11.30.3.1 bool Parma_Polyhedra_Library::Constraint_System::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 151 of file Constraint_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.30.3.2 const Constraint & Parma_Polyhedra_Library::Constraint_System::const_iterator::operator∗ () const [inline]

Dereference operator.

Definition at line 122 of file Constraint_System.inlines.hh.

11.30.3.3 Constraint_System::const_iterator Parma_Polyhedra_Library::Constraint_System::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Definition at line 139 of file Constraint_System.inlines.hh.

References operator++().

11.30.3.4 Constraint_System::const_iterator & Parma_Polyhedra_Library::Constraint_System::const_iterator::operator++ () [inline]

Prefix increment operator.

Definition at line 132 of file Constraint_System.inlines.hh.

References i, and skip_forward().

Referenced by operator++().

11.30.3.5 const Constraint ∗ Parma_Polyhedra_Library::Constraint_System::const_iterator::operator-> () const [inline]

Indirect member selector.

Definition at line 127 of file Constraint_System.inlines.hh.

References i.

11.30.3.6 Constraint_System::const_iterator & Parma_Polyhedra_Library::Constraint_System::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Definition at line 115 of file Constraint_System.inlines.hh.

References csp, Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.30.3.7 bool Parma_Polyhedra_Library::Constraint_System::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 146 of file Constraint_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.30.3.8 void Parma_Polyhedra_Library::Constraint_System::const_iterator::skip_forward () [private]

`*this` skips to the next non-trivial constraint.

Definition at line 324 of file Constraint_System.cc.

References csp, Parma_Polyhedra_Library::Matrix::end(), and i.

Referenced by operator++().

11.30.4 Friends And Related Function Documentation

11.30.4.1 friend class Constraint_System [friend]

Definition at line 248 of file Constraint_System.defs.hh.

11.30.5 Member Data Documentation

11.30.5.1 const Linear_System* Parma_Polyhedra_Library::Constraint_System::const_iterator::csp [private]

A const pointer to the matrix of constraints.

Definition at line 254 of file Constraint_System.defs.hh.

Referenced by operator=(), and skip_forward().

11.30.5.2 Linear_System::const_iterator Parma_Polyhedra_Library::Constraint_System::const_iterator::i [private]

The const iterator over the matrix of constraints.

Definition at line 251 of file Constraint_System.defs.hh.

Referenced by operator!=(), operator++(), operator->(), operator=(), operator==(), and skip_forward().

The documentation for this class was generated from the following files:

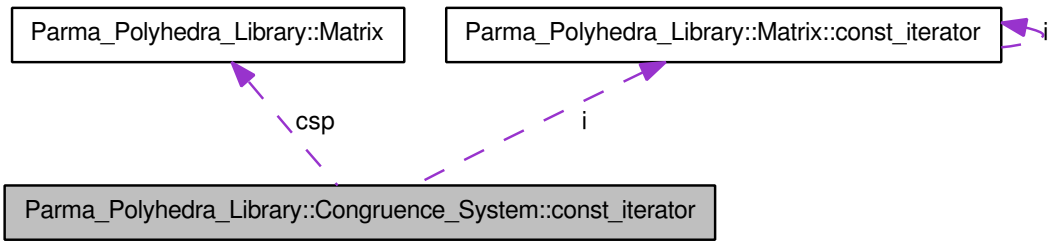
- Constraint_System.defs.hh
- Constraint_System.inlines.hh
- Constraint_System.cc

11.31 Parma_Polyhedra_Library::Congruence_System::const_iterator Class Reference

An iterator over a system of congruences.

`#include <Congruence_System.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Congruence_System::const_iterator:



Public Member Functions

- const_iterator ()
  Default constructor.
- const_iterator (const const_iterator &y)
  Ordinary copy constructor.
- ~const_iterator ()
  Destructor.
- const_iterator & operator= (const const_iterator &y)
  Assignment operator.
- const Congruence & operator* () const
  Dereference operator.
- const Congruence * operator-> () const

Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Member Functions

- const_iterator (const Matrix::const_iterator &iter, const Congruence_System &cgs)

  Constructor.

- void skip_forward ()

  `*this` *skips to the next non-trivial congruence.*

Private Attributes

- Matrix::const_iterator i

  The const iterator over the matrix of congruences.

- const Matrix * csp

  A const pointer to the matrix of congruences.

Friends

- class Congruence_System

11.31.1 Detailed Description

An iterator over a system of congruences. A const_iterator is used to provide read-only access to each congruence contained in an object of Congruence_System.

Example

The following code prints the system of congruences defining the grid `gr`:

```
const Congruence_System& cgs = gr.congruences();
for (Congruence_System::const_iterator i = cgs.begin(),
       cgs_end = cgs.end(); i != cgs_end; ++i)
  cout << *i << endl;
```

Definition at line 235 of file Congruence_System.defs.hh.

11.31.2 Constructor & Destructor Documentation

11.31.2.1 Parma_Polyhedra_Library::Congruence_System::const_iterator::const_iterator () [inline]

Default constructor.

Definition at line 114 of file Congruence_System.inlines.hh.

11.31.2.2 Parma_Polyhedra_Library::Congruence_System::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Definition at line 119 of file Congruence_System.inlines.hh.

11.31.2.3 Parma_Polyhedra_Library::Congruence_System::const_iterator::~const_iterator () [inline]

Destructor.

Definition at line 124 of file Congruence_System.inlines.hh.

11.31.2.4 Parma_Polyhedra_Library::Congruence_System::const_iterator::const_iterator (const Matrix::const_iterator & *iter*, const Congruence_System & *cgs*) [inline, private]

Constructor.

Definition at line 170 of file Congruence_System.inlines.hh.

11.31.3 Member Function Documentation

11.31.3.1 bool Parma_Polyhedra_Library::Congruence_System::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 164 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.31.3.2 const Congruence & Parma_Polyhedra_Library::Congruence_System::const_iterator::operator* () const [inline]

Dereference operator.

Definition at line 135 of file Congruence_System.inlines.hh.

11.31.3.3 Congruence_System::const_iterator Parma_Polyhedra_Library::Congruence_System::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Definition at line 152 of file Congruence_System.inlines.hh.

References operator++().

11.31.3.4 Congruence_System::const_iterator & Parma_Polyhedra_Library::Congruence_System::const_iterator::operator++ () [inline]

Prefix increment operator.

Definition at line 145 of file Congruence_System.inlines.hh.

References i, and skip_forward().

Referenced by operator++().

11.31.3.5 const Congruence * Parma_Polyhedra_Library::Congruence_System::const_iterator::operator-> () const [inline]

Indirect member selector.

Definition at line 140 of file Congruence_System.inlines.hh.

References i.

11.31.3.6 Congruence_System::const_iterator & Parma_Polyhedra_Library::Congruence_System::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Definition at line 128 of file Congruence_System.inlines.hh.

References csp, Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.31.3.7 bool Parma_Polyhedra_Library::Congruence_System::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 159 of file Congruence_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.31.3.8 void Parma_Polyhedra_Library::Congruence_System::const_iterator::skip_forward () [private]

`*this` skips to the next non-trivial congruence.

Definition at line 248 of file Congruence_System.cc.

References csp, Parma_Polyhedra_Library::Matrix::end(), and i.

Referenced by operator++().

11.31.4 Friends And Related Function Documentation

11.31.4.1 friend class Congruence_System [friend]

Definition at line 279 of file Congruence_System.defs.hh.

11.31.5 Member Data Documentation

11.31.5.1 const Matrix* Parma_Polyhedra_Library::Congruence_System::const_iterator::csp [private]

A const pointer to the matrix of congruences.

Definition at line 285 of file Congruence_System.defs.hh.

Referenced by operator=(), and skip_forward().

11.31.5.2 Matrix::const_iterator Parma_Polyhedra_Library::Congruence_System::const_iterator::i [private]

The const iterator over the matrix of congruences.

Definition at line 282 of file Congruence_System.defs.hh.

Referenced by operator!=(), operator++(), operator->(), operator=(), operator==(), and skip_forward().

The documentation for this class was generated from the following files:

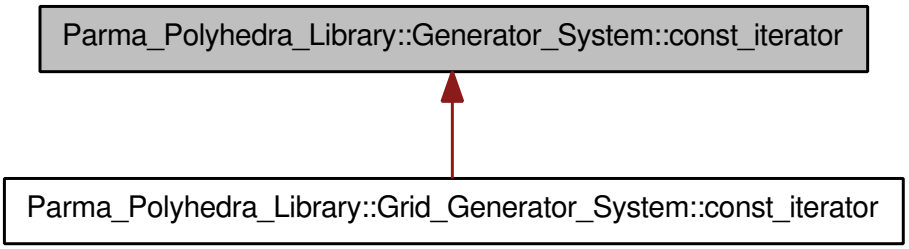
- Congruence_System.defs.hh
- Congruence_System.inlines.hh
- Congruence_System.cc

11.32 Parma_Polyhedra_Library::Generator_System::const_iterator Class Reference

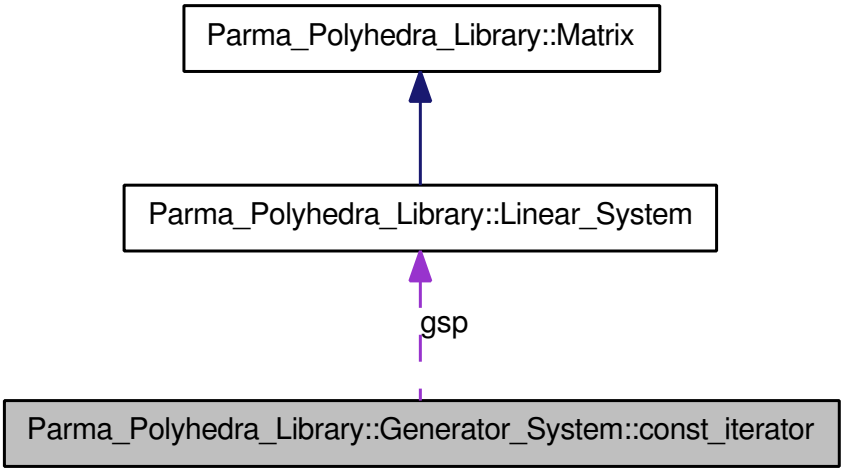
An iterator over a system of generators.

`#include <Generator_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Generator_System::const_iterator:



Collaboration diagram for Parma_Polyhedra_Library::Generator_System::const_iterator:



Public Member Functions

- const_iterator ()

  Default constructor.

- const_iterator (const const_iterator &y)

  Ordinary copy constructor.

- ~const_iterator ()

  Destructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- const Generator & operator∗ () const

  Dereference operator.

- const Generator ∗ operator-> () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Member Functions

- const_iterator (const Linear_System::const_iterator &iter, const Generator_System &gsys)

  Constructor.

- void skip_forward ()

  `*this` *skips to the next generator, skipping those closure points that are immediately followed by a matching point.*

Private Attributes

- Linear_System::const_iterator i

  The const iterator over the Linear_System.

- const Linear_System ∗ gsp

  A const pointer to the Linear_System.

Friends

- class Generator_System

11.32.1 Detailed Description

An iterator over a system of generators. A const_iterator is used to provide read-only access to each generator contained in an object of Generator_System.

Example

The following code prints the system of generators of the polyhedron `ph`:

```
const Generator_System& gs = ph.generators();
for (Generator_System::const_iterator i = gs.begin(),
       gs_end = gs.end(); i != gs_end; ++i)
  cout << *i << endl;
```

The same effect can be obtained more concisely by using more features of the STL:

```
const Generator_System& gs = ph.generators();
copy(gs.begin(), gs.end(), ostream_iterator<Generator>(cout, "\n"));
```

Definition at line 250 of file Generator_System.defs.hh.

11.32.2 Constructor & Destructor Documentation

11.32.2.1 Parma_Polyhedra_Library::Generator_System::const_iterator::const_iterator () [inline]

Default constructor.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 95 of file Generator_System.inlines.hh.

11.32.2.2 Parma_Polyhedra_Library::Generator_System::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 100 of file Generator_System.inlines.hh.

11.32.2.3 Parma_Polyhedra_Library::Generator_System::const_iterator::~const_iterator () [inline]

Destructor.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 105 of file Generator_System.inlines.hh.

11.32.2.4 Parma_Polyhedra_Library::Generator_System::const_iterator::const_iterator (const Linear_System::const_iterator & *iter*, const Generator_System & *gsys*) [inline, private]

Constructor.

Definition at line 153 of file Generator_System.inlines.hh.

11.32.3 Member Function Documentation

11.32.3.1 bool Parma_Polyhedra_Library::Generator_System::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 147 of file Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.32.3.2 const Generator & Parma_Polyhedra_Library::Generator_System::const_iterator::operator* () const [inline]

Dereference operator.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 117 of file Generator_System.inlines.hh.

11.32.3.3 Generator_System::const_iterator Parma_Polyhedra_Library::Generator_System::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 135 of file Generator_System.inlines.hh.

References operator++().

11.32.3.4 Generator_System::const_iterator & Parma_Polyhedra_Library::Generator_System::const_iterator::operator++ () [inline]

Prefix increment operator.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 127 of file Generator_System.inlines.hh.

References gsp, i, Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), and skip_forward().

Referenced by operator++().

11.32.3.5 const Generator * Parma_Polyhedra_Library::Generator_System::const_iterator::operator-> () const [inline]

Indirect member selector.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 122 of file Generator_System.inlines.hh.

References i.

11.32.3.6 Generator_System::const_iterator & Parma_Polyhedra_Library::Generator_System::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 110 of file Generator_System.inlines.hh.

References gsp, Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.32.3.7 bool Parma_Polyhedra_Library::Generator_System::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System::const_iterator.

Definition at line 142 of file Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::const_iterator::i, and i.

11.32.3.8 void Parma_Polyhedra_Library::Generator_System::const_iterator::skip_forward () [private]

`*this` skips to the next generator, skipping those closure points that are immediately followed by a matching point.

Definition at line 266 of file Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::end(), gsp, i, Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Generator::is_matching_closure_point(), and Parma_Polyhedra_Library::Generator::is_point().

Referenced by operator++().

11.32.4 Friends And Related Function Documentation

11.32.4.1 friend class Generator_System [friend]

Definition at line 294 of file Generator_System.defs.hh.

11.32.5 Member Data Documentation

11.32.5.1 const Linear_System* Parma_Polyhedra_Library::Generator_System::const_iterator::gsp [private]

A const pointer to the Linear_System.

Definition at line 300 of file Generator_System.defs.hh.

Referenced by operator++(), operator=(), and skip_forward().

11.32.5.2 Linear_System::const_iterator Parma_Polyhedra_Library::Generator_System::const_iterator::i [private]

The const iterator over the Linear_System.

Definition at line 297 of file Generator_System.defs.hh.

Referenced by operator!=(), operator++(), operator->(), operator=(), operator==(), and skip_forward().

The documentation for this class was generated from the following files:

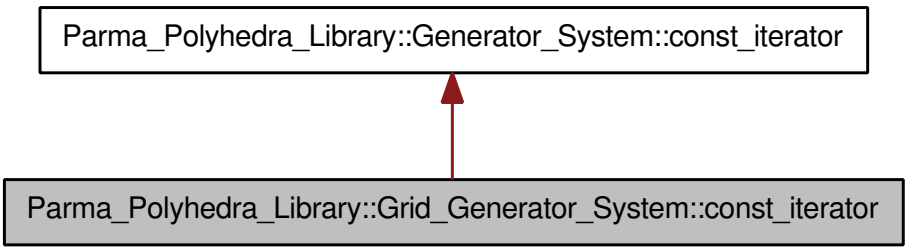
- Generator_System.defs.hh
- Generator_System.inlines.hh
- Generator_System.cc

11.33 Parma_Polyhedra_Library::Grid_Generator_System::const_iterator Class Reference

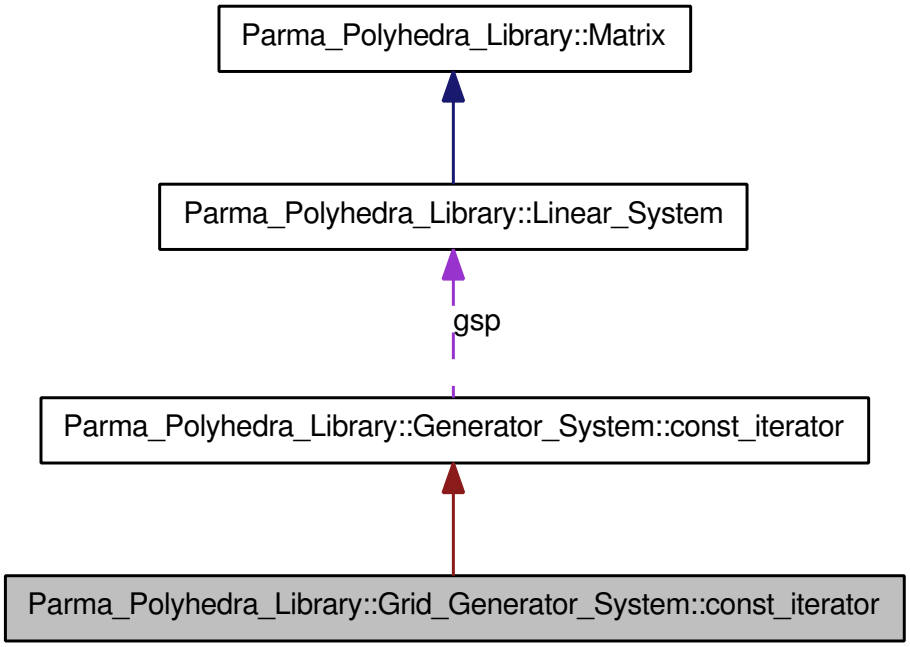
An iterator over a system of grid generators.

`#include <Grid_Generator_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Grid_Generator_System::const_iterator:



Collaboration diagram for Parma_Polyhedra_Library::Grid_Generator_System::const_iterator:



Public Member Functions

- const_iterator ()
  Default constructor.
- const_iterator (const const_iterator &y)
  Ordinary copy constructor.
- ~const_iterator ()
  Destructor.
- const_iterator & operator= (const const_iterator &y)
  Assignment operator.
- const Grid_Generator & operator* () const
  Dereference operator.
- const Grid_Generator * operator-> () const
  Indirect member selector.
- const_iterator & operator++ ()
  Prefix increment operator.
- const_iterator operator++ (int)
  Postfix increment operator.
- bool operator== (const const_iterator &y) const

Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Member Functions

- const_iterator (const Generator_System::const_iterator &y)

  Copy constructor from Generator_System::const_iterator.

Friends

- class Grid_Generator_System

11.33.1 Detailed Description

An iterator over a system of grid generators. A const_iterator is used to provide read-only access to each generator contained in an object of Grid_Generator_System.

Example

The following code prints the system of generators of the grid `gr`:

```
const Grid_Generator_System& ggs = gr.generators();
for (Grid_Generator_System::const_iterator i = ggs.begin(),
      ggs_end = ggs.end(); i != ggs_end; ++i)
  cout << *i << endl;
```

The same effect can be obtained more concisely by using more features of the STL:

```
const Grid_Generator_System& ggs = gr.generators();
copy(ggs.begin(), ggs.end(), ostream_iterator<Grid_Generator>(cout, "\n"));
```

Definition at line 267 of file Grid_Generator_System.defs.hh.

11.33.2 Constructor & Destructor Documentation

11.33.2.1 Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::const_iterator () [inline]

Default constructor.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 169 of file Grid_Generator_System.inlines.hh.

11.33.2.2 Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::const_iterator (const const_iterator & *y*) [inline]

Ordinary copy constructor.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 174 of file Grid_Generator_System.inlines.hh.

11.33.2.3 Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::~const_iterator () [inline]

Destructor.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 179 of file Grid_Generator_System.inlines.hh.

11.33.2.4 Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::const_iterator (const Generator_System::const_iterator & *y*) [inline, private]

Copy constructor from Generator_System::const_iterator.

Definition at line 233 of file Grid_Generator_System.inlines.hh.

11.33.3 Member Function Documentation

11.33.3.1 bool Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator!= (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 222 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Linear_System::operator!=().

11.33.3.2 const Grid_Generator & Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator* () const [inline]

Dereference operator.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 190 of file Grid_Generator_System.inlines.hh.

11.33.3.3 Grid_Generator_System::const_iterator Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator++ (int) [inline]

Postfix increment operator.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 208 of file Grid_Generator_System.inlines.hh.

References operator++().

11.33.3.4 Grid_Generator_System::const_iterator & Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator++ () [inline]

Prefix increment operator.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 202 of file Grid_Generator_System.inlines.hh.

Referenced by operator++().

11.33.3.5 const Grid_Generator * Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator-> () const [inline]

Indirect member selector.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 196 of file Grid_Generator_System.inlines.hh.

11.33.3.6 Grid_Generator_System::const_iterator & Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator= (const const_iterator & *y*) [inline]

Assignment operator.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 184 of file Grid_Generator_System.inlines.hh.

11.33.3.7 bool Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator== (const const_iterator & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Reimplemented from Parma_Polyhedra_Library::Generator_System::const_iterator.

Definition at line 216 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Grid_Generator_System::operator==.

11.33.4 Friends And Related Function Documentation

11.33.4.1 friend class Grid_Generator_System [friend]

Definition at line 312 of file Grid_Generator_System.defs.hh.

The documentation for this class was generated from the following files:

- Grid_Generator_System.defs.hh
- Grid_Generator_System.inlines.hh

11.34 Parma_Polyhedra_Library::const_iterator_to_const< Container > Class Template Reference

A const_iterator on a sequence of read-only objects.

`#include <iterator_to_const.defs.hh>`

Public Types

- typedef Traits::iterator_category iterator_category
- typedef Traits::value_type value_type
- typedef Traits::difference_type difference_type
- typedef Traits::pointer pointer
- typedef Traits::reference reference

Public Member Functions

- const_iterator_to_const ()

  Default constructor.

- const_iterator_to_const (const const_iterator_to_const &y)

  Copy constructor.

- const_iterator_to_const (const iterator_to_const< Container > &y)

  Constructs from the corresponding non-const iterator.

- reference operator* () const

  Dereference operator.

- pointer operator-> () const

  Indirect member selector.

- const_iterator_to_const & operator++ ()

  Prefix increment operator.

- const_iterator_to_const operator++ (int)

  Postfix increment operator.

- const_iterator_to_const & operator-- ()

  Prefix decrement operator.

- const_iterator_to_const operator-- (int)

  Postfix decrement operator.

- bool operator== (const const_iterator_to_const &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const const_iterator_to_const &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Types

- typedef Container::const_iterator Base

  The type of the underlying const_iterator.

- typedef std::iterator_traits< Base > Traits

  A shortcut for naming traits.

Private Member Functions

- const_iterator_to_const (const Base &b)

  Constructs from the lower-level const_iterator.

Private Attributes

- Base base

  A const_iterator on the sequence of elements.

Friends

- class iterator_to_const< Container >
- class Powerset

Related Functions

(Note that these are not member functions.)

- template<typename Container >
  bool operator== (const iterator_to_const< Container > &x, const const_iterator_to_const< Container > &y)

  Mixed comparison operator: returns `true` *if and only if (the const version of)* `x` *is identical to* `y`.

- template<typename Container >
  bool operator!= (const iterator_to_const< Container > &x, const const_iterator_to_const< Container > &y)

  Mixed comparison operator: returns `true` *if and only if (the const version of)* `x` *is different from* `y`*.*

11.34.1 Detailed Description

template<typename Container> class Parma_Polyhedra_Library::const_iterator_to_const< Container >

A const_iterator on a sequence of read-only objects. This class, besides implementing a read-only bidirectional iterator on a read-only sequence of objects, ensures interoperability with template class iterator_to_const.

Definition at line 125 of file iterator_to_const.defs.hh.

11.34.2 Member Typedef Documentation

11.34.2.1 template<typename Container> typedef Container::const_iterator Parma_Polyhedra_Library::const_iterator_to_const< Container >::Base [private]

The type of the underlying const_iterator.

Definition at line 128 of file iterator_to_const.defs.hh.

11.34.2.2 template<typename Container> typedef Traits::difference_type Parma_Polyhedra_Library::const_iterator_to_const< Container >::difference_type

Definition at line 147 of file iterator_to_const.defs.hh.

11.34.2.3 template<typename Container> typedef Traits::iterator_category Parma_Polyhedra_Library::const_iterator_to_const< Container >::iterator_category

Definition at line 145 of file iterator_to_const.defs.hh.

11.34.2.4 template<typename Container> typedef Traits::pointer Parma_Polyhedra_Library::const_iterator_to_const< Container >::pointer

Definition at line 148 of file iterator_to_const.defs.hh.

11.34.2.5 template<typename Container> typedef Traits::reference Parma_Polyhedra_Library::const_iterator_to_const< Container >::reference

Definition at line 149 of file iterator_to_const.defs.hh.

11.34.2.6 template<typename Container> typedef std::iterator_traits<Base> Parma_Polyhedra_Library::const_iterator_to_const< Container >::Traits [private]

A shortcut for naming traits.

Definition at line 131 of file iterator_to_const.defs.hh.

11.34.2.7 template<typename Container> typedef Traits::value_type Parma_Polyhedra_Library::const_iterator_to_const< Container >::value_type

Definition at line 146 of file iterator_to_const.defs.hh.

11.34.3 Constructor & Destructor Documentation

11.34.3.1 template<typename Container > Parma_Polyhedra_Library::const_iterator_to_const< Container >::const_iterator_to_const (const Base & *b*) [inline, private]

Constructs from the lower-level const_iterator.

Definition at line 117 of file iterator_to_const.inlines.hh.

11.34.3.2 template<typename Container > Parma_Polyhedra_Library::const_iterator_to_const< Container >::const_iterator_to_const () [inline]

Default constructor.

Definition at line 104 of file iterator_to_const.inlines.hh.

11.34.3.3 template<typename Container > Parma_Polyhedra_Library::const_iterator_to_const< Container >::const_iterator_to_const (const const_iterator_to_const< Container > & *y*) [inline]

Copy constructor.

Definition at line 111 of file iterator_to_const.inlines.hh.

11.34.3.4 template<typename Container> Parma_Polyhedra_Library::const_iterator_to_const< Container >::const_iterator_to_const (const iterator_to_const< Container > & *y*) [inline]

Constructs from the corresponding non-const iterator.

Definition at line 180 of file iterator_to_const.inlines.hh.

11.34.4 Member Function Documentation

11.34.4.1 template<typename Container > bool Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator!= (const const_iterator_to_const< Container > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 173 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::operator==().

11.34.4.2 template<typename Container > const_iterator_to_const< Container >::reference Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator* () const [inline]

Dereference operator.

Definition at line 123 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::base.

11.34.4.3 template<typename Container > const_iterator_to_const< Container > Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator++ (int) [inline]

Postfix increment operator.

Definition at line 142 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator++().

11.34.4.4 template<typename Container > const_iterator_to_const< Container > & Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator++ () [inline]

Prefix increment operator.

Definition at line 135 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::base.

Referenced by Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator++().

11.34.4.5 template<typename Container > const_iterator_to_const< Container > Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator-- (int) [inline]

Postfix decrement operator.

Definition at line 157 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator--().

11.34.4.6 template<typename Container > const_iterator_to_const< Container > & Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator-- () [inline]

Prefix decrement operator.

Definition at line 150 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::base.

Referenced by Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator--().

11.34.4.7 template<typename Container > const_iterator_to_const< Container >::pointer Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator-> () const [inline]

Indirect member selector.

Definition at line 129 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::base.

11.34.4.8 template<typename Container > bool Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator== (const const_iterator_to_const< Container > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 166 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::const_iterator_to_const< Container >::base.

11.34.5 Friends And Related Function Documentation

11.34.5.1 template<typename Container> friend class iterator_to_const< Container > [friend]

Definition at line 139 of file iterator_to_const.defs.hh.

11.34.5.2 template<typename Container > bool operator!= (const iterator_to_const< Container > & *x*, const const_iterator_to_const< Container > & *y*) [related]

Mixed comparison operator: returns `true` if and only if (the const version of) `x` is different from `y`.

Definition at line 195 of file iterator_to_const.inlines.hh.

11.34.5.3 template<typename Container > bool operator== (const iterator_to_const< Container > & *x*, const const_iterator_to_const< Container > & *y*) [related]

Mixed comparison operator: returns `true` if and only if (the const version of) `x` is identical to `y`.

Definition at line 187 of file iterator_to_const.inlines.hh.

11.34.5.4 template<typename Container> friend class Powerset [friend]

Definition at line 140 of file iterator_to_const.defs.hh.

11.34.6 Member Data Documentation

11.34.6.1 template<typename Container> Base Parma_Polyhedra_Library::const_iterator_to_const< Container >::base [private]

A const_iterator on the sequence of elements.

Definition at line 134 of file iterator_to_const.defs.hh.

Referenced by Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator∗(), Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator++(), Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator--(), Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator->(), and Parma_Polyhedra_Library::const_iterator_to_const< Container >::operator==().

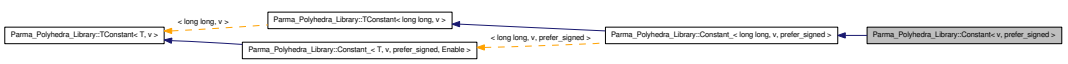
The documentation for this class was generated from the following files:

- iterator_to_const.defs.hh
- iterator_to_const.inlines.hh

11.35 Parma_Polyhedra_Library::Constant< v, prefer_signed > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constant< v, prefer_signed >:



Collaboration diagram for Parma_Polyhedra_Library::Constant< v, prefer_signed >:

11.35.1 Detailed Description

template<long long v, bool prefer_signed = true> struct Parma_Polyhedra_Library::Constant< v, prefer_signed >

Definition at line 381 of file globals.defs.hh.

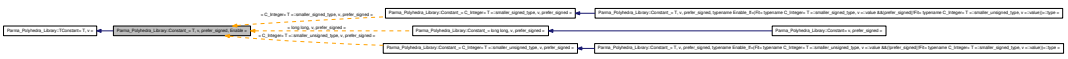
The documentation for this struct was generated from the following file:

- globals.defs.hh

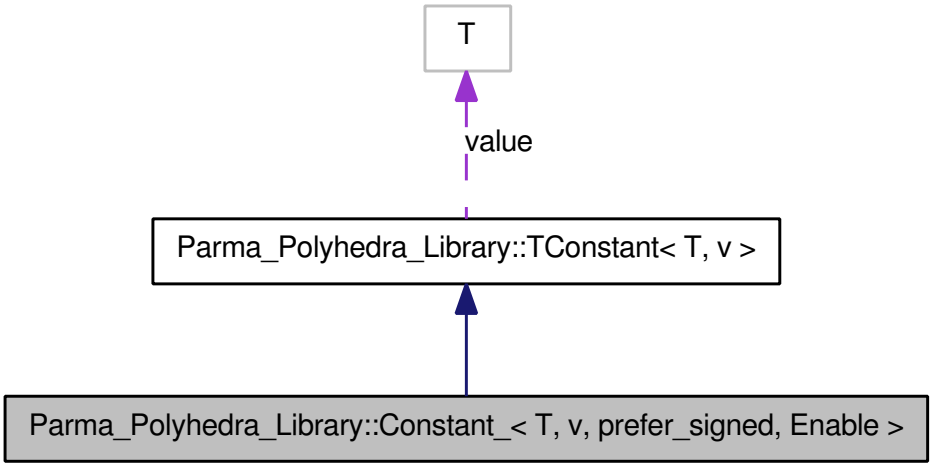
11.36 Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable >:



Collaboration diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable >:



11.36.1 Detailed Description

template<typename T, long long v, bool prefer_signed = true, typename Enable = void> struct Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable >

Definition at line 361 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

- globals.defs.hh

11.37 Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type >:

Collaboration diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type >:

11.37.1 Detailed Description

template<typename T, long long v, bool prefer_signed> struct Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type >

Definition at line 365 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

- globals.defs.hh

11.38 Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type >:

Collaboration diagram for Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_-If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type >:

11.38.1 Detailed Description

template<typename T, long long v, bool prefer_signed> struct Parma_Polyhedra_-Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type >

Definition at line 373 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

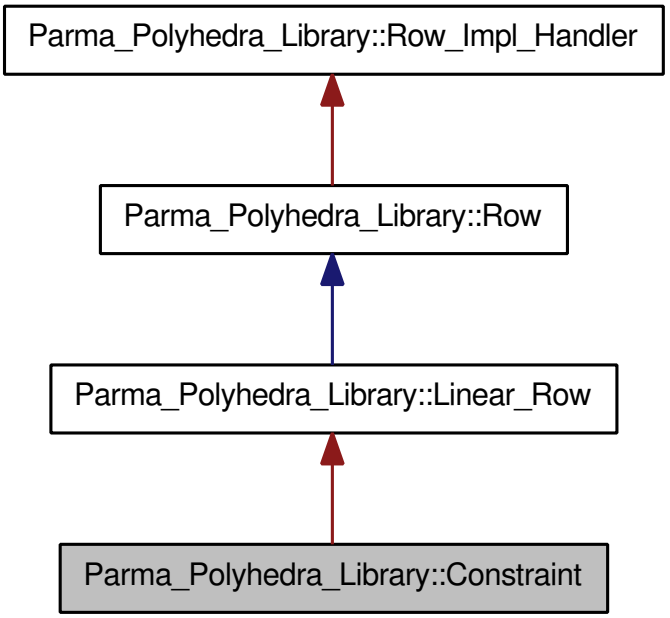
- globals.defs.hh

11.39 Parma_Polyhedra_Library::Constraint Class Reference

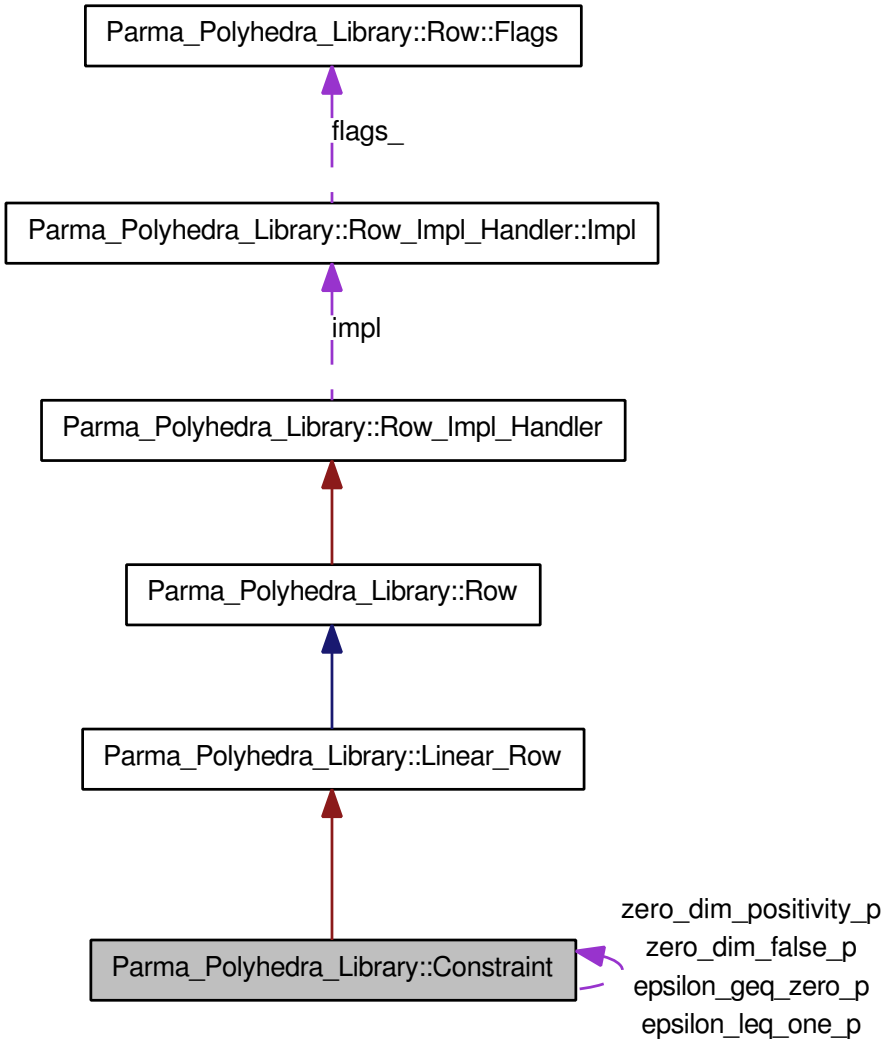
A linear equality or inequality.

`#include <Constraint.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constraint:



Collaboration diagram for Parma_Polyhedra_Library::Constraint:



Public Types

- enum Type { EQUALITY, NONSTRICT_INEQUALITY, STRICT_INEQUALITY }

  The constraint type.

Public Member Functions

- Constraint (const Constraint &c)

  Ordinary copy constructor.

- Constraint (const Congruence &cg)

  Copy-constructs from equality congruence `cg`.

- ~Constraint ()

Destructor.

- Constraint & operator= (const Constraint &c)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Type type () const

  Returns the constraint type of `*this`.

- bool is_equality () const

  Returns `true` *if and only if* `*this` *is an equality constraint.*

- bool is_inequality () const

  Returns `true` *if and only if* `*this` *is an inequality constraint (either strict or non-strict).*

- bool is_nonstrict_inequality () const

  Returns `true` *if and only if* `*this` *is a non-strict inequality constraint.*

- bool is_strict_inequality () const

  Returns `true` *if and only if* `*this` *is a strict inequality constraint.*

- Coefficient_traits::const_reference coefficient (Variable v) const

  Returns the coefficient of `v` *in* `*this`.

- Coefficient_traits::const_reference inhomogeneous_term () const

  Returns the inhomogeneous term of `*this`.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool is_tautological () const

  Returns `true` *if and only if* `*this` *is a tautology (i.e., an always true constraint).*

- bool is_inconsistent () const

  Returns `true` *if and only if* `*this` *is inconsistent (i.e., an always false constraint).*

- bool is_equivalent_to (const Constraint &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are equivalent constraints.*

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void swap (Constraint &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Constraint can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Constraint & zero_dim_false ()

  The unsatisfiable (zero-dimension space) constraint $0 = 1$.

- static const Constraint & zero_dim_positivity ()

  The true (zero-dimension space) constraint $0 \leq 1$, *also known as positivity constraint.*

Private Member Functions

- friend Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Constraint &c)
- Constraint ()

  Default constructor: private and not implemented.

- Constraint (Linear_Expression &e, Type type, Topology topology)

  Builds a constraint of type `type` *and topology* `topology`, *stealing the coefficients from* `e`.

- Constraint (const Congruence &cg, dimension_type sz, dimension_type capacity)

  Constructs from a congruence, with specified size and capacity.

- void throw_invalid_argument (const char *method, const char *message) const

  Throws a `std::invalid_argument` *exception containing error message* `message`.

- void throw_dimension_incompatible (const char *method, const char *name_var, Variable v) const

  Throws a `std::invalid_argument` *exception containing the appropriate error message.*

- Constraint (const Constraint &c, dimension_type sz)

  Copy constructor with given size.

- void set_is_equality ()

  Sets the constraint type to `EQUALITY`.

- void set_is_inequality ()

  Sets the constraint to be an inequality.

Static Private Member Functions

- static Constraint construct_epsilon_geq_zero ()

  Builds a new copy of the zero-dimension space constraint $\epsilon \geq 0$ *(used to implement NNC polyhedra).*

- static const Constraint & epsilon_geq_zero ()

  Returns the zero-dimension space constraint $\epsilon \geq 0$.

- static const Constraint & epsilon_leq_one ()

  The zero-dimension space constraint $\epsilon \leq 1$ *(used to implement NNC polyhedra).*

Static Private Attributes

- static const Constraint * zero_dim_false_p = 0

  Holds (between class initialization and finalization) a pointer to the unsatisfiable (zero-dimension space) constraint $0 = 1$.

- static const Constraint * zero_dim_positivity_p = 0

  Holds (between class initialization and finalization) a pointer to the true (zero-dimension space) constraint $0 \leq 1$, *also known as positivity constraint.*

- static const Constraint * epsilon_geq_zero_p = 0

  Holds (between class initialization and finalization) a pointer to the zero-dimension space constraint $\epsilon \geq 0$.

- static const Constraint * epsilon_leq_one_p = 0

  Holds (between class initialization and finalization) a pointer to the zero-dimension space constraint $\epsilon \leq 1$ *(used to implement NNC polyhedra).*

Friends

- class Parma_Polyhedra_Library::Congruence
- class Parma_Polyhedra_Library::Scalar_Products
- class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign
- class Parma_Polyhedra_Library::Constraint_System
- class Parma_Polyhedra_Library::Constraint_System::const_iterator
- class Parma_Polyhedra_Library::Polyhedron
- Constraint operator== (const Linear_Expression &e1, const Linear_Expression &e2)

Returns the constraint `e1 = e2`.

- Constraint operator== (Variable v1, Variable v2)

  Returns the constraint `v1 = v2`.

- Constraint operator== (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the constraint `e = n`.

- Constraint operator== (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the constraint `n = e`.

- Constraint operator>= (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the constraint `e1 >= e2`.

- Constraint operator>= (Variable v1, Variable v2)

  Returns the constraint `v1 >= v2`.

- Constraint operator>= (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the constraint `e >= n`.

- Constraint operator>= (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the constraint `n >= e`.

- Constraint operator<= (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the constraint `e1 <= e2`.

- Constraint operator<= (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the constraint `e <= n`.

- Constraint operator<= (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the constraint `n <= e`.

- Constraint operator> (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the constraint `e1 > e2`.

- Constraint operator> (Variable v1, Variable v2)

  Returns the constraint `v1 > v2`.

- Constraint operator> (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the constraint `e > n`.

- Constraint operator> (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the constraint `n > e`.

- Constraint operator< (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the constraint `e1 < e2`.

- Constraint operator< (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the constraint `e < n`.

- Constraint operator< (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the constraint `n < e`.

Related Functions

(Note that these are not member functions.)

- bool operator== (const Constraint &x, const Constraint &y)

  Returns `true` *if and only if* `x` *is equivalent to* `y`.

- bool operator!= (const Constraint &x, const Constraint &y)

  Returns `true` *if and only if* `x` *is not equivalent to* `y`.

- Constraint operator<= (Variable v1, Variable v2)

  Returns the constraint `v1 <= v2`.

- Constraint operator< (Variable v1, Variable v2)

  Returns the constraint `v1 < v2`.

- void swap (Parma_Polyhedra_Library::Constraint &x, Parma_Polyhedra_Library::Constraint &y)

  Specializes `std::swap`.

- std::ostream & operator<< (std::ostream &s, const Constraint &c)

  Output operator.

- std::ostream & operator<< (std::ostream &s, const Constraint::Type &t)

  Output operator.

11.39.1 Detailed Description

A linear equality or inequality. An object of the class Constraint is either:

- an equality: $\sum_{i=0}^{n-1} a_i x_i + b = 0$;
- a non-strict inequality: $\sum_{i=0}^{n-1} a_i x_i + b \geq 0$; or
- a strict inequality: $\sum_{i=0}^{n-1} a_i x_i + b > 0$;

where n is the dimension of the space, a_i is the integer coefficient of variable x_i and b is the integer inhomogeneous term.

How to build a constraint

Constraints are typically built by applying a relation symbol to a pair of linear expressions. Available relation symbols are equality (==), non-strict inequalities (>= and <=) and strict inequalities (< and >). The space dimension of a constraint is defined as the maximum space dimension of the arguments of its constructor.

In the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds the equality constraint $3x + 5y - z = 0$, having space dimension 3:

```
Constraint eq_c(3*x + 5*y - z == 0);
```

The following code builds the (non-strict) inequality constraint $4x \geq 2y - 13$, having space dimension 2:

```
Constraint ineq_c(4*x >= 2*y - 13);
```

The corresponding strict inequality constraint $4x > 2y - 13$ is obtained as follows:

```
Constraint strict_ineq_c(4*x > 2*y - 13);
```

An unsatisfiable constraint on the zero-dimension space $\mathbb{R}^0$ can be specified as follows:

```
Constraint false_c = Constraint::zero_dim_false();
```

Equivalent, but more involved ways are the following:

```
Constraint false_c1(Linear_Expression::zero() == 1);
Constraint false_c2(Linear_Expression::zero() >= 1);
Constraint false_c3(Linear_Expression::zero() > 0);
```

In contrast, the following code defines an unsatisfiable constraint having space dimension 3:

```
Constraint false_c(0*z == 1);
```

How to inspect a constraint

Several methods are provided to examine a constraint and extract all the encoded information: its space dimension, its type (equality, non-strict inequality, strict inequality) and the value of its integer coefficients.

Example 2

The following code shows how it is possible to access each single coefficient of a constraint. Given an inequality constraint (in this case $x - 5y + 3z \leq 4$), we construct a new constraint corresponding to its complement (thus, in this case we want to obtain the strict inequality constraint $x - 5y + 3z > 4$).

```
Constraint c1(x - 5*y + 3*z <= 4);
cout << "Constraint c1: " << c1 << endl;
if (c1.is_equality())
  cout << "Constraint c1 is not an inequality." << endl;
else {
  Linear_Expression e;
  for (dimension_type i = c1.space_dimension(); i-- > 0; )
    e += c1.coefficient(Variable(i)) * Variable(i);
  e += c1.inhomogeneous_term();
  Constraint c2 = c1.is_strict_inequality() ? (e <= 0) : (e < 0);
  cout << "Complement c2: " << c2 << endl;
}
```

The actual output is the following:

```
Constraint c1: -A + 5*B - 3*C >= -4
Complement c2: A - 5*B + 3*C > 4
```

Note that, in general, the particular output obtained can be syntactically different from the (semantically equivalent) constraint considered.

Definition at line 261 of file Constraint.defs.hh.

11.39.2 Member Enumeration Documentation

11.39.2.1 enum Parma_Polyhedra_Library::Constraint::Type

The constraint type.

Enumerator:

EQUALITY The constraint is an equality.
NONSTRICT_INEQUALITY The constraint is a non-strict inequality.
STRICT_INEQUALITY The constraint is a strict inequality.

Definition at line 286 of file Constraint.defs.hh.

11.39.3 Constructor & Destructor Documentation

11.39.3.1 Parma_Polyhedra_Library::Constraint::Constraint (const Constraint & *c*) [inline]

Ordinary copy constructor.

Definition at line 41 of file Constraint.inlines.hh.

11.39.3.2 Parma_Polyhedra_Library::Constraint::Constraint (const Congruence & *cg*) [explicit]

Copy-constructs from equality congruence `cg`.

Exceptions

std::invalid_argument Thrown if `cg` is a proper congruence.

Definition at line 62 of file Constraint.cc.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Linear_Row::strong_normalize().

11.39.3.3 Parma_Polyhedra_Library::Constraint::~Constraint () [inline]

Destructor.

Definition at line 51 of file Constraint.inlines.hh.

11.39.3.4 Parma_Polyhedra_Library::Constraint::Constraint () [private]

Default constructor: private and not implemented.

Referenced by initialize().

11.39.3.5 Parma_Polyhedra_Library::Constraint::Constraint (Linear_Expression & *e*, Type *type*, Topology *topology*) [inline, private]

Builds a constraint of type `type` and topology `topology`, stealing the coefficients from `e`.

Definition at line 32 of file Constraint.inlines.hh.

References EQUALITY, Parma_Polyhedra_Library::Linear_Row::flags(), Parma_Polyhedra_Library::Linear_Row::LINE_OR_EQUALITY, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, STRICT_INEQUALITY, and swap().

11.39.3.6 Parma_Polyhedra_Library::Constraint::Constraint (const Congruence & *cg*, dimension_type *sz*, dimension_type *capacity*) [private]

Constructs from a congruence, with specified size and capacity.

Definition at line 80 of file Constraint.cc.

11.39.3.7 Parma_Polyhedra_Library::Constraint::Constraint (const Constraint & *c*, dimension_type *sz*) [inline, private]

Copy constructor with given size.

Definition at line 46 of file Constraint.inlines.hh.

11.39.4 Member Function Documentation

11.39.4.1 void Parma_Polyhedra_Library::Constraint::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 353 of file Constraint.inlines.hh.

References ascii_dump().

11.39.4.2 void Parma_Polyhedra_Library::Constraint::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Referenced by ascii_dump().

11.39.4.3 bool Parma_Polyhedra_Library::Constraint::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 358 of file Constraint.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Problem::ascii_load(), and Parma_Polyhedra_Library::MIP_Problem::ascii_load().

11.39.4.4 Coefficient_traits::const_reference Parma_Polyhedra_Library::Constraint::coefficient (Variable *v*) const [inline]

Returns the coefficient of `v` in `*this`.

Exceptions

std::invalid_argument thrown if the index of `v` is greater than or equal to the space dimension of `*this`.

Definition at line 113 of file Constraint.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), operator<<(), Parma_Polyhedra_Library::MIP_Problem::parse_constraints(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::PIP_Problem::solve(), Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.39.4.5 PPL::Constraint Parma_Polyhedra_Library::Constraint::construct_epsilon_geq_zero () [static, private]

Builds a new copy of the zero-dimension space constraint $\epsilon \geq 0$ (used to implement NNC polyhedra).

Definition at line 56 of file Constraint.cc.

References NONSTRICT_INEQUALITY, and Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED.

Referenced by initialize().

11.39.4.6 const Constraint & Parma_Polyhedra_Library::Constraint::epsilon_geq_zero () [inline, static, private]

Returns the zero-dimension space constraint $\epsilon \geq 0$.

Definition at line 341 of file Constraint.inlines.hh.

References epsilon_geq_zero_p.

Referenced by Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints().

11.39.4.7 const Constraint & Parma_Polyhedra_Library::Constraint::epsilon_leq_one () [inline, static, private]

The zero-dimension space constraint $\epsilon \leq 1$ (used to implement NNC polyhedra).

Definition at line 347 of file Constraint.inlines.hh.

References epsilon_leq_one_p.

Referenced by Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.39.4.8 memory_size_type Parma_Polyhedra_Library::Constraint::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 125 of file Constraint.inlines.hh.

11.39.4.9 void Parma_Polyhedra_Library::Constraint::finalize () [static]

Finalizes the class.

Definition at line 245 of file Constraint.cc.

References epsilon_geq_zero_p, epsilon_leq_one_p, zero_dim_false_p, and zero_dim_positivity_p.

11.39.4.10 Coefficient_traits::const_reference Parma_Polyhedra_Library::Constraint::inhomogeneous_term () const [inline]

Returns the inhomogeneous term of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 120 of file Constraint.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), operator<<(), Parma_Polyhedra_Library::MIP_Problem::parse_constraints(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::PIP_Problem::solve(), Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.39.4.11 void Parma_Polyhedra_Library::Constraint::initialize () [static]

Initializes the class.

Definition at line 226 of file Constraint.cc.

References Parma_Polyhedra_Library::Coefficient_one(), Constraint(), construct_epsilon_geq_zero(), epsilon_geq_zero_p, epsilon_leq_one_p, Parma_Polyhedra_Library::Linear_Expression::zero(), zero_dim_false_p, and zero_dim_positivity_p.

11.39.4.12 bool Parma_Polyhedra_Library::Constraint::is_equality () const [inline]

Returns `true` if and only if `*this` is an equality constraint.

Definition at line 71 of file Constraint.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Grid::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), is_inconsistent(), is_tautological(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), Parma_Polyhedra_Library::MIP_Problem::parse_constraints(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Polyhedron::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Grid::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_

Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::PIP_Problem::solve(), type(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.39.4.13 bool Parma_Polyhedra_Library::Constraint::is_equivalent_to (const Constraint & *y*) const

Returns `true` if and only if `*this` and `y` are equivalent constraints.

Constraints having different space dimensions are not equivalent. Note that constraints having different types may nonetheless be equivalent, if they both are tautologies or inconsistent.

Definition at line 179 of file Constraint.cc.

References is_inconsistent(), is_tautological(), Parma_Polyhedra_Library::Row::normalize(), space_dimension(), STRICT_INEQUALITY, and type().

Referenced by operator!=(), and operator==().

11.39.4.14 bool Parma_Polyhedra_Library::Constraint::is_inconsistent () const

Returns `true` if and only if `*this` is inconsistent (i.e., an always false constraint).

An inconsistent constraint can have either one of the following forms:

- an equality: $\sum_{i=0}^{n-1} 0x_i + b = 0$, where $b \neq 0$; or
- a non-strict inequality: $\sum_{i=0}^{n-1} 0x_i + b \geq 0$, where $b < 0$; or
- a strict inequality: $\sum_{i=0}^{n-1} 0x_i + b > 0$, where $b \leq 0$.

Definition at line 139 of file Constraint.cc.

References Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), is_equality(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Grid::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), is_equivalent_to(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::Polyhedron::refine_no_check(), Parma_Polyhedra_Library::Grid::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.39.4.15 bool Parma_Polyhedra_Library::Constraint::is_inequality () const [inline]

Returns `true` if and only if `*this` is an inequality constraint (either strict or non-strict).

Definition at line 76 of file Constraint.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality().

Referenced by Parma_Polyhedra_Library::Grid::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), Parma_Polyhedra_Library::MIP_Problem::is_satisfied(), Parma_Polyhedra_Library::MIP_Problem::parse_constraints(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), and Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign().

11.39.4.16 bool Parma_Polyhedra_Library::Constraint::is_nonstrict_inequality () const [inline]

Returns `true` if and only if `*this` is a non-strict inequality constraint.

Definition at line 93 of file Constraint.inlines.hh.

References NONSTRICT_INEQUALITY, and type().

Referenced by Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.39.4.17 bool Parma_Polyhedra_Library::Constraint::is_strict_inequality () const [inline]

Returns `true` if and only if `*this` is a strict inequality constraint.

Definition at line 98 of file Constraint.inlines.hh.

References STRICT_INEQUALITY, and type().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition_aux(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::PIP_Problem::solve(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.39.4.18 bool Parma_Polyhedra_Library::Constraint::is_tautological () const

Returns `true` if and only if `*this` is a tautology (i.e., an always true constraint).

A tautology can have either one of the following forms:

- an equality: $\sum_{i=0}^{n-1} 0x_i + 0 = 0$; or
- a non-strict inequality: $\sum_{i=0}^{n-1} 0x_i + b \geq 0$, where $b \geq 0$; or
- a strict inequality: $\sum_{i=0}^{n-1} 0x_i + b > 0$, where $b > 0$.

Definition at line 98 of file Constraint.cc.

References Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), is_equality(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Grid::add_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), is_equivalent_to(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.39.4.19 dimension_type Parma_Polyhedra_Library::Constraint::max_space_dimension () [inline, static]

Returns the maximum space dimension a Constraint can handle.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 61 of file Constraint.inlines.hh.

11.39.4.20 bool Parma_Polyhedra_Library::Constraint::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 329 of file Constraint.cc.

References Parma_Polyhedra_Library::Linear_Row::OK(), and Parma_Polyhedra_Library::Linear_Row::strong_normalize().

11.39.4.21 Constraint & Parma_Polyhedra_Library::Constraint::operator= (const Constraint & *c*) [inline]

Assignment operator.

Definition at line 55 of file Constraint.inlines.hh.

11.39.4.22 Parma_Polyhedra_Library::Constraint::Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Constraint & *c*) [private]

11.39.4.23 void Parma_Polyhedra_Library::Constraint::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

11.39.4.24 void Parma_Polyhedra_Library::Constraint::set_is_equality () [inline, private]

Sets the constraint type to `EQUALITY`.

Definition at line 103 of file Constraint.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), and Parma_Polyhedra_Library::Polyhedron::concatenate_assign().

11.39.4.25 void Parma_Polyhedra_Library::Constraint::set_is_inequality () [inline, private]

Sets the constraint to be an inequality.

Whether the constraint type will become `NONSTRICT_INEQUALITY` or `STRICT_INEQUALITY` depends on the topology and the value of the low-level coefficients of the constraint.

Definition at line 108 of file Constraint.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::set_is_ray_or_point_or_inequality().

11.39.4.26 dimension_type Parma_Polyhedra_Library::Constraint::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 66 of file Constraint.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::PIP_Problem::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_-

constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::Grid::add_constraint(), Parma_Polyhedra_Library::Box< ITV >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Grid::add_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), coefficient(), Parma_Polyhedra_Library::BD_Shape< T >::extract_bounded_difference(), Parma_Polyhedra_Library::Box< ITV >::extract_interval_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), is_equivalent_to(), Parma_Polyhedra_Library::MIP_Problem::is_satisfied(), Parma_Polyhedra_Library::MIP_Problem::is_saturated(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator()(), operator<<(), Parma_Polyhedra_Library::MIP_Problem::parse_constraints(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Grid::refine_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::Grid::refine_with_constraint(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::PIP_Problem::solve(), Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.39.4.27 void Parma_Polyhedra_Library::Constraint::swap (Constraint & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 363 of file Constraint.inlines.hh.

Referenced by Constraint(), and swap().

11.39.4.28 void Parma_Polyhedra_Library::Constraint::throw_dimension_incompatible (const char * *method*, const char * *name_var*, Variable *v*) const [private]

Throws a `std::invalid_argument` exception containing the appropriate error message.

Definition at line 45 of file Constraint.cc.

References Parma_Polyhedra_Library::Variable::space_dimension(), and space_dimension().

Referenced by coefficient().

11.39.4.29 void Parma_Polyhedra_Library::Constraint::throw_invalid_argument (const char * *method*, const char * *message*) const [private]

Throws a `std::invalid_argument` exception containing error message `message`.

Definition at line 36 of file Constraint.cc.

11.39.4.30 memory_size_type Parma_Polyhedra_Library::Constraint::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 130 of file Constraint.inlines.hh.

11.39.4.31 Constraint::Type Parma_Polyhedra_Library::Constraint::type () const [inline]

Returns the constraint type of `*this`.

Definition at line 81 of file Constraint.inlines.hh.

References EQUALITY, is_equality(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Row::size(), and STRICT_INEQUALITY.

Referenced by Parma_Polyhedra_Library::Box< ITV >::add_constraint_no_check(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), is_equivalent_to(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), is_nonstrict_inequality(), is_strict_inequality(), operator<<(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraint_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.39.4.32 const Constraint & Parma_Polyhedra_Library::Constraint::zero_dim_false () [inline, static]

The unsatisfiable (zero-dimension space) constraint $0 = 1$.

Definition at line 329 of file Constraint.inlines.hh.

References zero_dim_false_p.

Referenced by Parma_Polyhedra_Library::Constraint_System::initialize().

11.39.4.33 const Constraint & Parma_Polyhedra_Library::Constraint::zero_dim_positivity () [inline, static]

The true (zero-dimension space) constraint $0 \leq 1$, also known as *positivity constraint*.

Definition at line 335 of file Constraint.inlines.hh.

References zero_dim_positivity_p.

Referenced by Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), Parma_Polyhedra_Library::PIP_Problem::ascii_load(), and Parma_Polyhedra_Library::MIP_Problem::ascii_load().

11.39.5 Friends And Related Function Documentation

11.39.5.1 bool operator!= (const Constraint & *x*, const Constraint & *y*) [related]

Returns `true` if and only if `x` is not equivalent to `y`.

Definition at line 142 of file Constraint.inlines.hh.

References is_equivalent_to().

11.39.5.2 Constraint operator< (Variable *v1*, Variable *v2*) [related]

Returns the constraint `v1` $<$ `v2`.

Definition at line 312 of file Constraint.inlines.hh.

11.39.5.3 Constraint operator< (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the constraint `n` $<$ `e`.

Definition at line 318 of file Constraint.inlines.hh.

11.39.5.4 Constraint operator< (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the constraint `e` $<$ `n`.

Definition at line 324 of file Constraint.inlines.hh.

11.39.5.5 Constraint operator< (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the constraint `e1 < e2`.

Definition at line 306 of file Constraint.inlines.hh.

11.39.5.6 std::ostream & operator<< (std::ostream & *s*, const Constraint::Type & *t*) [related]

Output operator.

Definition at line 309 of file Constraint.cc.

References EQUALITY, NONSTRICT_INEQUALITY, and STRICT_INEQUALITY.

11.39.5.7 std::ostream & operator<< (std::ostream & *s*, const Constraint & *c*) [related]

Output operator.

Definition at line 265 of file Constraint.cc.

References coefficient(), Parma_Polyhedra_Library::Coefficient_zero(), EQUALITY, inhomogeneous_term(), Parma_Polyhedra_Library::neg_assign(), NONSTRICT_INEQUALITY, PPL_DIRTY_TEMP_COEFFICIENT, space_dimension(), STRICT_INEQUALITY, and type().

11.39.5.8 Constraint operator<= (Variable *v1*, Variable *v2*) [related]

Returns the constraint `v1 <= v2`.

Definition at line 288 of file Constraint.inlines.hh.

11.39.5.9 Constraint operator<= (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the constraint `n <= e`.

Definition at line 294 of file Constraint.inlines.hh.

11.39.5.10 Constraint operator<= (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the constraint `e <= n`.

Definition at line 300 of file Constraint.inlines.hh.

11.39.5.11 Constraint operator<= (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the constraint `e1 <= e2`.

Definition at line 282 of file Constraint.inlines.hh.

11.39.5.12 bool operator== (const Constraint & *x*, const Constraint & *y*) [related]

Returns `true` if and only if `x` is equivalent to `y`.

Definition at line 136 of file Constraint.inlines.hh.

References is_equivalent_to().

11.39.5.13 Constraint operator== (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the constraint `n = e`.

Definition at line 212 of file Constraint.inlines.hh.

11.39.5.14 Constraint operator== (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the constraint `e = n`.

Definition at line 246 of file Constraint.inlines.hh.

11.39.5.15 Constraint operator== (Variable *v1*, Variable *v2*) [friend]

Returns the constraint `v1 = v2`.

Definition at line 158 of file Constraint.inlines.hh.

11.39.5.16 Constraint operator== (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the constraint `e1 = e2`.

Definition at line 148 of file Constraint.inlines.hh.

11.39.5.17 Constraint operator> (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the constraint `n > e`.

Definition at line 232 of file Constraint.inlines.hh.

11.39.5.18 Constraint operator> (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the constraint `e > n`.

Definition at line 266 of file Constraint.inlines.hh.

11.39.5.19 Constraint operator> (Variable *v1*, Variable *v2*) [friend]

Returns the constraint `v1 > v2`.

Definition at line 202 of file Constraint.inlines.hh.

11.39.5.20 Constraint operator> (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the constraint `e1 > e2`.

Definition at line 183 of file Constraint.inlines.hh.

11.39.5.21 Constraint operator>= (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the constraint `n >= e`.

Definition at line 222 of file Constraint.inlines.hh.

11.39.5.22 Constraint operator>= (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the constraint `e >= n`.

Definition at line 256 of file Constraint.inlines.hh.

11.39.5.23 Constraint operator>= (Variable *v1*, Variable *v2*) [friend]

Returns the constraint `v1 >= v2`.

Definition at line 176 of file Constraint.inlines.hh.

11.39.5.24 Constraint operator>= (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the constraint `e1 >= e2`.

Definition at line 166 of file Constraint.inlines.hh.

11.39.5.25 friend class Parma_Polyhedra_Library::Congruence [friend]

Definition at line 435 of file Constraint.defs.hh.

11.39.5.26 friend class Parma_Polyhedra_Library::Constraint_System [friend]

Definition at line 438 of file Constraint.defs.hh.

11.39.5.27 friend class Parma_Polyhedra_Library::Constraint_System::const_iterator [friend]

Definition at line 439 of file Constraint.defs.hh.

11.39.5.28 friend class Parma_Polyhedra_Library::Polyhedron [friend]

Definition at line 441 of file Constraint.defs.hh.

11.39.5.29 friend class Parma_Polyhedra_Library::Scalar_Products [friend]

Definition at line 436 of file Constraint.defs.hh.

11.39.5.30 friend class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign [friend]

Definition at line 437 of file Constraint.defs.hh.

11.39.5.31 void swap (Parma_Polyhedra_Library::Constraint & *x*, Parma_Polyhedra_Library::Constraint & *y*) [related]

Specializes `std::swap`.

Definition at line 373 of file Constraint.inlines.hh.

References swap().

11.39.6 Member Data Documentation

11.39.6.1 const PPL::Constraint * Parma_Polyhedra_Library::Constraint::epsilon_geq_zero_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the zero-dimension space constraint $\epsilon \geq 0$.

Definition at line 426 of file Constraint.defs.hh.

Referenced by epsilon_geq_zero(), finalize(), and initialize().

11.39.6.2 const PPL::Constraint * Parma_Polyhedra_Library::Constraint::epsilon_leq_one_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the zero-dimension space constraint $\epsilon \leq 1$ (used to implement NNC polyhedra).

Definition at line 433 of file Constraint.defs.hh.

Referenced by epsilon_leq_one(), finalize(), and initialize().

11.39.6.3 const PPL::Constraint * Parma_Polyhedra_Library::Constraint::zero_dim_false_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the unsatisfiable (zero-dimension space) constraint $0 = 1$.

Definition at line 413 of file Constraint.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_false().

11.39.6.4 const PPL::Constraint * Parma_Polyhedra_Library::Constraint::zero_dim_positivity_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the true (zero-dimension space) constraint $0 \leq 1$, also known as *positivity constraint*.

Definition at line 420 of file Constraint.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_positivity().

The documentation for this class was generated from the following files:

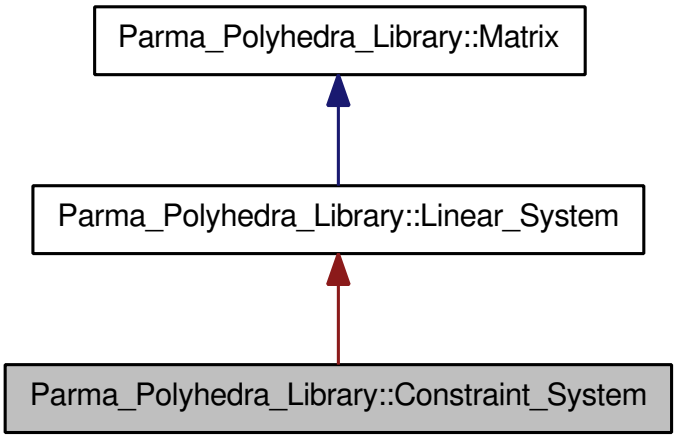
- Constraint.defs.hh
- Constraint.inlines.hh
- Constraint.cc

11.40 Parma_Polyhedra_Library::Constraint_System Class Reference

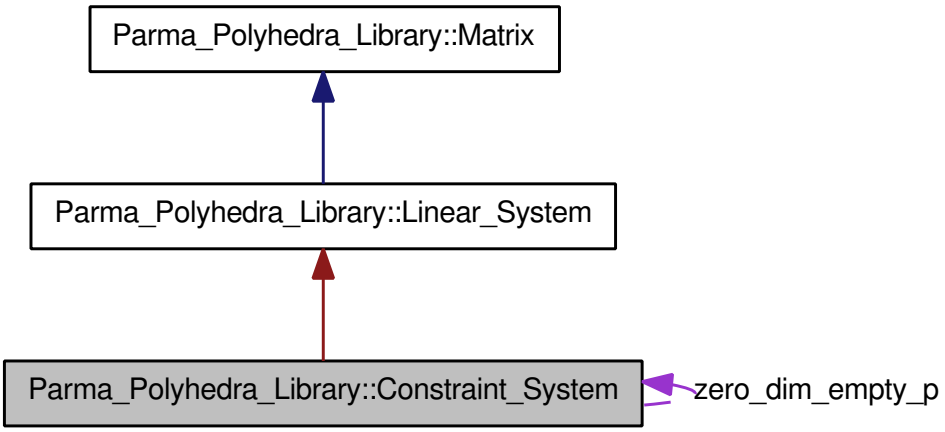
A system of constraints.

`#include <Constraint_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Constraint_System:



Collaboration diagram for Parma_Polyhedra_Library::Constraint_System:



Classes

- class const_iterator

  An iterator over a system of constraints.

Public Member Functions

- Constraint_System ()

  Default constructor: builds an empty system of constraints.

- Constraint_System (const Constraint &c)

  Builds the singleton system containing only constraint `c`.

- Constraint_System (const Congruence_System &cgs)

  Builds a system containing copies of any equalities in `cgs`.

- Constraint_System (const Constraint_System &cs)

  Ordinary copy constructor.

- ~Constraint_System ()

  Destructor.

- Constraint_System & operator= (const Constraint_System &y)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- bool has_equalities () const

  Returns `true` *if and only if* `*this` *contains one or more equality constraints.*

- bool has_strict_inequalities () const

  Returns `true` *if and only if* `*this` *contains one or more strict inequality constraints.*

- void clear ()

  Removes all the constraints from the constraint system and sets its space dimension to 0.

- void insert (const Constraint &c)

  Inserts in `*this` *a copy of the constraint* `c`, *increasing the number of space dimensions if needed.*

- bool empty () const

  Returns `true` *if and only if* `*this` *has no constraints.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first constraint, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void swap (Constraint_System &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Constraint_System can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Constraint_System & zero_dim_empty ()

  Returns the singleton system containing only Constraint::zero_dim_false().

Private Member Functions

- Constraint_System (Topology topol)

  Builds an empty system of constraints having the specified topology.

- Constraint_System (Topology topol, dimension_type n_rows, dimension_type n_columns)

  Builds a system of `n_rows` *constraints on a* `n_columns` *- 1 dimensional space (including the* ϵ *dimension, if* `topol` *is* `NOT_NECESSARILY_CLOSED`*).*

- bool adjust_topology_and_space_dimension (Topology topol, dimension_type num_dimensions)

  Adjusts `*this` *so that it matches the topology and the number of space dimensions given as parameters (adding or removing columns if needed). Returns* `false` *if and only if* `topol` *is equal to* `NECESSARILY_CLOSED` *and* `*this` *contains strict inequalities.*

- Constraint & operator[ ] (dimension_type k)

  Returns the `k`− th constraint of the system.

- const Constraint & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k`− th constraint of the system.

- bool satisfies_all_constraints (const Generator &g) const

  Returns `true` if `g` satisfies all the constraints.

- void affine_preimage (dimension_type v, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator)

  Substitutes a given column of coefficients by a given affine expression.

- dimension_type num_equalities () const

  Returns the number of equality constraints.

- dimension_type num_inequalities () const

  Returns the number of inequality constraints.

- void simplify ()

  Applies Gaussian elimination and back-substitution so as to provide a partial simplification of the system of constraints.

- void insert_pending (const Constraint &c)

  Inserts in `*this` a copy of the constraint `c`, increasing the number of space dimensions if needed. It is a pending constraint.

- void add_low_level_constraints ()

  Adds low-level constraints to the constraint system.

Static Private Attributes

- static const Constraint_System * zero_dim_empty_p = 0

  Holds (between class initialization and finalization) a pointer to the singleton system containing only Constraint::zero_dim_false().

Friends

- class const_iterator
- class Parma_Polyhedra_Library::Polyhedron
- bool operator== (const Polyhedron &x, const Polyhedron &y)

  Returns `true` if and only if `x` and `y` are the same polyhedron.

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Constraint_System &cs)

  Output operator.

- void swap (Parma_Polyhedra_Library::Constraint_System &x, Parma_Polyhedra_Library::Constraint_System &y)

  Specializes `std::swap`.

11.40.1 Detailed Description

A system of constraints. An object of the class Constraint_System is a system of constraints, i.e., a multiset of objects of the class Constraint. When inserting constraints in a system, space dimensions are automatically adjusted so that all the constraints in the system are defined on the same vector space.

In all the examples it is assumed that variables `x` and `y` are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a system of constraints corresponding to a square in $\mathbb{R}^2$:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
```

Note that: the constraint system is created with space dimension zero; the first and third constraint insertions increase the space dimension to 1 and 2, respectively.

Example 2

By adding four strict inequalities to the constraint system of the previous example, we can remove just the four vertices from the square defined above.

```
cs.insert(x + y > 0);
cs.insert(x + y < 6);
cs.insert(x - y < 3);
cs.insert(y - x < 3);
```

Example 3

The following code builds a system of constraints corresponding to a half-strip in $\mathbb{R}^2$:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x - y <= 0);
cs.insert(x - y + 1 >= 0);
```

Note

After inserting a multiset of constraints in a constraint system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* constraint system will be available, where original constraints may have been reordered, removed (if they are trivial, duplicate or implied by other constraints), linearly combined, etc.

Definition at line 128 of file Constraint_System.defs.hh.

11.40.2 Constructor & Destructor Documentation

11.40.2.1 Parma_Polyhedra_Library::Constraint_System::Constraint_System () [inline]

Default constructor: builds an empty system of constraints.

Definition at line 32 of file Constraint_System.inlines.hh.

Referenced by initialize().

11.40.2.2 Parma_Polyhedra_Library::Constraint_System::Constraint_System (const Constraint & *c*) [inline, explicit]

Builds the singleton system containing only constraint `c`.

Definition at line 37 of file Constraint_System.inlines.hh.

References insert().

11.40.2.3 Parma_Polyhedra_Library::Constraint_System::Constraint_System (const Congruence_System & *cgs*) [explicit]

Builds a system containing copies of any equalities in `cgs`.

Definition at line 40 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), and insert().

11.40.2.4 Parma_Polyhedra_Library::Constraint_System::Constraint_System (const Constraint_System & *cs*) [inline]

Ordinary copy constructor.

Definition at line 43 of file Constraint_System.inlines.hh.

11.40.2.5 Parma_Polyhedra_Library::Constraint_System::~Constraint_System () [inline]

Destructor.

Definition at line 60 of file Constraint_System.inlines.hh.

11.40.2.6 Parma_Polyhedra_Library::Constraint_System::Constraint_System (Topology *topol*) [inline, explicit, private]

Builds an empty system of constraints having the specified topology.

Definition at line 48 of file Constraint_System.inlines.hh.

11.40.2.7 Parma_Polyhedra_Library::Constraint_System::Constraint_System (Topology *topol*, dimension_type *n_rows*, dimension_type *n_columns*) [inline, private]

Builds a system of `n_rows` constraints on a `n_columns` - 1 dimensional space (including the ϵ dimension, if `topol` is `NOT_NECESSARILY_CLOSED`).

Definition at line 53 of file Constraint_System.inlines.hh.

11.40.3 Member Function Documentation

11.40.3.1 void Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints () [inline, private]

Adds low-level constraints to the constraint system.

Definition at line 181 of file Constraint_System.inlines.hh.

References Parma_Polyhedra_Library::Constraint::epsilon_geq_zero(), Parma_Polyhedra_Library::Constraint::epsilon_leq_one(), insert(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), and Parma_Polyhedra_Library::Constraint::zero_dim_positivity().

Referenced by Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.40.3.2 bool Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension (Topology *topol*, dimension_type *num_dimensions*) [private]

Adjusts `*this` so that it matches the topology and the number of space dimensions given as parameters (adding or removing columns if needed). Returns `false` if and only if `topol` is equal to `NECESSARILY_CLOSED` and `*this` contains strict inequalities.

Definition at line 51 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), has_strict_inequalities(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row(), Parma_Polyhedra_Library::Linear_System::set_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), Parma_Polyhedra_

Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_System::sort_rows(), space_-dimension(), Parma_Polyhedra_Library::swap(), swap(), Parma_Polyhedra_Library::Matrix::swap_-columns(), Parma_Polyhedra_Library::Linear_System::topology(), and Parma_Polyhedra_-Library::Linear_System::unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_-Library::Polyhedron::constraints(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.40.3.3 void Parma_Polyhedra_Library::Constraint_System::affine_preimage (dimension_type *v*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator*) [private]

Substitutes a given column of coefficients by a given affine expression.

Parameters

v Index of the column to which the affine transformation is substituted.

expr The numerator of the affine transformation: $\sum_{i=0}^{n-1} a_i x_i + b$;

denominator The denominator of the affine transformation.

We want to allow affine transformations (see Section Images and Preimages of Affine Transfer Relations) having any rational coefficients. Since the coefficients of the constraints are integers we must also provide an integer `denominator` that will be used as denominator of the affine transformation. The denominator is required to be a positive integer.

The affine transformation substitutes the matrix of constraints by a new matrix whose elements a'_{ij} are built from the old one a_{ij} as follows:

$$a'_{ij} = \begin{cases} a_{ij} * \text{denominator} + a_{iv} * \text{expr}[j] & \text{for } j \neq v; \\ \text{expr}[v] * a_{iv} & \text{for } j = v. \end{cases}$$

`expr` is a constant parameter and unaltered by this computation.

Definition at line 424 of file Constraint_System.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Matrix::num_-columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), space_dimension(), and Parma_-Polyhedra_Library::Linear_System::strong_normalize().

Referenced by Parma_Polyhedra_Library::Polyhedron::affine_image(), and Parma_Polyhedra_-Library::Polyhedron::affine_preimage().

11.40.3.4 void Parma_Polyhedra_Library::Constraint_System::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 479 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, and Parma_Polyhedra_Library::Constraint::type().

11.40.3.5 void Parma_Polyhedra_Library::Constraint_System::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Referenced by Parma_Polyhedra_Library::Polyhedron::ascii_dump(), Parma_Polyhedra_Library::PIP_Tree_Node::ascii_dump(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.40.3.6 bool Parma_Polyhedra_Library::Constraint_System::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 514 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), and Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::ascii_load().

11.40.3.7 Constraint_System::const_iterator Parma_Polyhedra_Library::Constraint_System::begin () const [inline]

Returns the const_iterator pointing to the first constraint, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 163 of file Constraint_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::begin().

Referenced by Parma_Polyhedra_Library::PIP_Problem::add_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::Grid::add_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_

Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), Parma_Polyhedra_Library::Congruence_System::Congruence_System(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), empty(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::MIP_Problem::MIP_Problem(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::PIP_Decision_Node::OK(), Parma_Polyhedra_Library::PIP_Tree_Node::OK(), operator<<(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::PIP_Tree_Node::print_tree(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Grid::refine_with_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_Decision_Node::solve(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.40.3.8 void Parma_Polyhedra_Library::Constraint_System::clear () [inline]

Removes all the constraints from the constraint system and sets its space dimension to 0.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 90 of file Constraint_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::set_empty(), and Parma_Polyhedra_Library::Polyhedron::set_zero_dim_univ().

11.40.3.9 bool Parma_Polyhedra_Library::Constraint_System::empty () const [inline]

Returns `true` if and only if `*this` has no constraints.

Definition at line 176 of file Constraint_System.inlines.hh.

References begin(), and end().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::PIP_Solution_Node::print_tree(), Parma_Polyhedra_Library::PIP_Tree_-

Node::print_tree(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.40.3.10 Constraint_System::const_iterator Parma_Polyhedra_Library::Constraint_System::end () const [inline]

Returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 170 of file Constraint_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::end().

Referenced by Parma_Polyhedra_Library::PIP_Problem::add_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::Grid::add_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), Parma_Polyhedra_Library::Congruence_System::Congruence_System(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), empty(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::MIP_Problem::MIP_Problem(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::PIP_Decision_Node::OK(), Parma_Polyhedra_Library::PIP_Tree_Node::OK(), operator<<(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::PIP_Tree_Node::print_tree(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Grid::refine_with_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_Decision_Node::solve(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.40.3.11 memory_size_type Parma_Polyhedra_Library::Constraint_System::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 198 of file Constraint_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::external_memory_in_bytes(), and Parma_Polyhedra_Library::PIP_Tree_Node::external_memory_in_bytes().

11.40.3.12 void Parma_Polyhedra_Library::Constraint_System::finalize () [static]

Finalizes the class.

Definition at line 596 of file Constraint_System.cc.

References zero_dim_empty_p.

11.40.3.13 bool Parma_Polyhedra_Library::Constraint_System::has_equalities () const

Returns `true` if and only if `*this` contains one or more equality constraints.

Definition at line 210 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation().

11.40.3.14 bool Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities () const

Returns `true` if and only if `*this` contains one or more strict inequality constraints.

Definition at line 220 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::MIP_Problem::MIP_Problem(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.40.3.15 void Parma_Polyhedra_Library::Constraint_System::initialize () [static]

Initializes the class.

Definition at line 589 of file Constraint_System.cc.

References Constraint_System(), zero_dim_empty_p, and Parma_Polyhedra_Library::Constraint::zero_dim_false().

11.40.3.16 void Parma_Polyhedra_Library::Constraint_System::insert (const Constraint & *c*)

Inserts in `*this` a copy of the constraint `c`, increasing the number of space dimensions if needed.

Definition at line 240 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), OK(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Linear_Row::topology(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruences(), add_low_level_constraints(), Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Constraint_System(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Polyhedron::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruences(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Polyhedron::select_CH78_constraints(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_Decision_Node::solve(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.40.3.17 void Parma_Polyhedra_Library::Constraint_System::insert_pending (const Constraint & *c*) [private]

Inserts in `*this` a copy of the constraint `c`, increasing the number of space dimensions if needed. It is a pending constraint.

Definition at line 271 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), OK(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Linear_Row::topology(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Polyhedron::refine_no_check(), and Parma_Polyhedra_Library::Polyhedron::refine_with_constraints().

11.40.3.18 dimension_type Parma_Polyhedra_Library::Constraint_System::max_space_dimension () [inline, static]

Returns the maximum space dimension a Constraint_System can handle.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 80 of file Constraint_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::max_space_dimension().

11.40.3.19 PPL::dimension_type Parma_Polyhedra_Library::Constraint_System::num_equalities () const [private]

Returns the number of equality constraints.

Definition at line 316 of file Constraint_System.cc.

References num_inequalities(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::H79_widening_assign(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Polyhedron::quick_equivalence_test(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators().

11.40.3.20 PPL::dimension_type Parma_Polyhedra_Library::Constraint_System::num_inequalities () const [private]

Returns the number of inequality constraints.

Definition at line 297 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by num_equalities().

11.40.3.21 bool Parma_Polyhedra_Library::Constraint_System::OK () const

Checks if all the invariants are satisfied.

Returns `true` if and only if `*this` is a valid Linear_System and each row in the system is a valid Constraint.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 603 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), and Parma_Polyhedra_Library::Matrix::OK().

Referenced by adjust_topology_and_space_dimension(), insert(), insert_pending(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.40.3.22 Constraint_System & Parma_Polyhedra_Library::Constraint_System::operator= (const Constraint_System & *y*) `[inline]`

Assignment operator.

Definition at line 64 of file Constraint_System.inlines.hh.

11.40.3.23 const Constraint & Parma_Polyhedra_Library::Constraint_System::operator[ ] (dimension_type *k*) const `[inline, private]`

Returns a constant reference to the `k`− th constraint of the system.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 75 of file Constraint_System.inlines.hh.

References operator[ ]().

11.40.3.24 Constraint & Parma_Polyhedra_Library::Constraint_System::operator[ ] (dimension_type *k*) `[inline, private]`

Returns the `k`− th constraint of the system.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 70 of file Constraint_System.inlines.hh.

Referenced by operator[ ]().

11.40.3.25 void Parma_Polyhedra_Library::Constraint_System::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

11.40.3.26 bool Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints (const Generator & *g*) const [private]

Returns `true` if `g` satisfies all the constraints.

Definition at line 331 of file Constraint_System.cc.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, space_dimension(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, Parma_Polyhedra_Library::Constraint::type(), and Parma_Polyhedra_Library::Generator::type().

Referenced by Parma_Polyhedra_Library::Polyhedron::relation_with().

11.40.3.27 void Parma_Polyhedra_Library::Constraint_System::simplify () [inline, private]

Applies Gaussian elimination and back-substitution so as to provide a partial simplification of the system of constraints.

It is assumed that the system has no pending constraints.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 208 of file Constraint_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::OK(), and Parma_Polyhedra_Library::Polyhedron::simplified_constraints().

11.40.3.28 dimension_type Parma_Polyhedra_Library::Constraint_System::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 85 of file Constraint_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::Grid::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::add_constraints(), Parma_Polyhedra_Library::Box< ITV >::add_constraints_no_check(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), adjust_topology_and_space_dimension(), affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Grid::Grid(), insert(), insert_pending(), Parma_Polyhedra_Library::Octagonal_Shape< T

>::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_-BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_-extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_-extrapolation_assign(), Parma_Polyhedra_Library::Box< ITV >::limited_CC76_extrapolation_-assign(), Parma_Polyhedra_Library::BD_Shape< T >::limited_CC76_extrapolation_assign(), Parma_-Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::MIP_-Problem::MIP_Problem(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_Library::Box< ITV >::refine_no_-check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Grid::refine_-with_constraints(), Parma_Polyhedra_Library::Box< ITV >::refine_with_constraints(), Parma_-Polyhedra_Library::BD_Shape< T >::refine_with_constraints(), satisfies_all_constraints(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), Parma_Polyhedra_-Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_-dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_-assign().

11.40.3.29 void Parma_Polyhedra_Library::Constraint_System::swap (Constraint_System & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 193 of file Constraint_System.inlines.hh.

Referenced by adjust_topology_and_space_dimension(), Parma_Polyhedra_-Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_-assign(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_-Decision_Node::solve(), and swap().

11.40.3.30 memory_size_type Parma_Polyhedra_Library::Constraint_System::total_memory_-in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 203 of file Constraint_System.inlines.hh.

11.40.3.31 const Constraint_System & Parma_Polyhedra_Library::Constraint_System::zero_-dim_empty () [inline, static]

Returns the singleton system containing only Constraint::zero_dim_false().

Definition at line 95 of file Constraint_System.inlines.hh.

References zero_dim_empty_p.

Referenced by Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Box< ITV >::minimized_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints().

11.40.4 Friends And Related Function Documentation

11.40.4.1 friend class const_iterator [friend]

Definition at line 311 of file Constraint_System.defs.hh.

11.40.4.2 std::ostream & operator<< (std::ostream & *s*, const Constraint_System & *cs*) [related]

Output operator.

Writes `true` if `cs` is empty. Otherwise, writes on `s` the constraints of `cs`, all in one row and separated by ", ".

Definition at line 622 of file Constraint_System.cc.

References begin(), and end().

11.40.4.3 bool operator== (const Polyhedron & *x*, const Polyhedron & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same polyhedron.

Note that `x` and `y` may be topology- and/or dimension-incompatible polyhedra: in those cases, the value `false` is returned.

Definition at line 3613 of file Polyhedron_public.cc.

11.40.4.4 friend class Parma_Polyhedra_Library::Polyhedron [friend]

Definition at line 312 of file Constraint_System.defs.hh.

11.40.4.5 void swap (Parma_Polyhedra_Library::Constraint_System & *x*, Parma_Polyhedra_Library::Constraint_System & *y*) [related]

Specializes `std::swap`.

Definition at line 219 of file Constraint_System.inlines.hh.

References swap().

11.40.5 Member Data Documentation

11.40.5.1 const PPL::Constraint_System * Parma_Polyhedra_Library::Constraint_System::zero_dim_empty_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the singleton system containing only Constraint::zero_dim_false().

Definition at line 309 of file Constraint_System.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_empty().

The documentation for this class was generated from the following files:

- Constraint_System.defs.hh
- Constraint_System.inlines.hh
- Constraint_System.cc
- Generator_System.defs.hh
- Polyhedron.defs.hh
- Polyhedron_public.cc

11.41 Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 > Class Template Reference

This class provides the reduction method for the Constraints_Product domain.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- Constraints_Reduction ()

  Default constructor.

- void product_reduce (D1 &d1, D2 &d2)

  The constraints reduction operator for sharing constraints between the domains.

- ~Constraints_Reduction ()

  Destructor.

11.41.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >

This class provides the reduction method for the Constraints_Product domain. The reduction classes are used to instantiate the Partially_Reduced_Product domain. This class adds the constraints defining each of the component domains to the other component.

Definition at line 135 of file Partially_Reduced_Product.defs.hh.

11.41.2 Constructor & Destructor Documentation

11.41.2.1 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >::Constraints_Reduction () [inline]

Default constructor.

Definition at line 771 of file Partially_Reduced_Product.inlines.hh.

11.41.2.2 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >::~Constraints_Reduction () [inline]

Destructor.

Definition at line 776 of file Partially_Reduced_Product.inlines.hh.

11.41.3 Member Function Documentation

11.41.3.1 template<typename D1 , typename D2 > void Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >::product_reduce (D1 & *d1*, D2 & *d2*) [inline]

The constraints reduction operator for sharing constraints between the domains.

The minimized constraint system defining the domain element `d1` is added to `d2` and the minimized constraint system defining `d2` is added to `d1`. In each case, the donor domain must provide a constraint system in minimal form; this must define a polyhedron in which the donor element is contained. The recipient domain selects a subset of these constraints that it can add to the recipient element. For example: if the domain `D1` is the Grid domain and `D2` the NNC Polyhedron domain, then only the equality constraints are copied from `d1` to `d2` and from `d2` to `d1`.

Parameters

d1 A pointset domain element;

d2 A pointset domain element;

Definition at line 468 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >::product_reduce(), and Parma_Polyhedra_Library::swap().

Referenced by Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce().

The documentation for this class was generated from the following files:

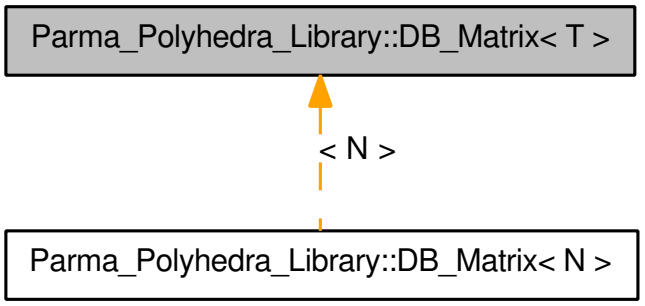
- Partially_Reduced_Product.defs.hh
- Partially_Reduced_Product.inlines.hh
- Partially_Reduced_Product.templates.hh

11.42 Parma_Polyhedra_Library::DB_Matrix< T > Class Template Reference

The base class for the square matrices.

`#include <DB_Matrix.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::DB_Matrix< T >:



Classes

- class const_iterator

  A read-only iterator over the rows of the matrix.

Public Member Functions

- DB_Matrix ()

  Builds an empty matrix.

- DB_Matrix (dimension_type n_rows)

  Builds a square matrix having the specified dimension.

- DB_Matrix (const DB_Matrix &y)

  Copy constructor.

- template<typename U >
  DB_Matrix (const DB_Matrix< U > &y)

  Constructs a conservative approximation of `y`.

- ~DB_Matrix ()

  Destructor.

- DB_Matrix & operator= (const DB_Matrix &y)

  Assignment operator.

- const_iterator begin () const

  Returns the const_iterator pointing to the first row, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

Returns the past-the-end const_iterator.

- void swap (DB_Matrix &y)

  Swaps `*this` with `y`.

- void grow (dimension_type new_n_rows)

  Makes the matrix grow by adding more rows and more columns.

- void resize_no_copy (dimension_type new_n_rows)

  Resizes the matrix without worrying about the old contents.

- dimension_type num_rows () const

  Returns the number of rows in the matrix.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

Subscript operators.

- DB_Row< T > & operator[ ] (dimension_type k)

  Returns a reference to the `k-th` row of the matrix.

- const DB_Row< T > & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k-th` row of the matrix.

Static Public Member Functions

- static dimension_type max_num_rows ()

  Returns the maximum number of rows a DB_Matrix can handle.

- static dimension_type max_num_columns ()

  Returns the maximum number of columns a DB_Matrix can handle.

Private Attributes

- std::vector< DB_Row< T > > rows

  The rows of the matrix.

- dimension_type row_size

  Size of the initialized part of each row.

- dimension_type row_capacity

  Capacity allocated for each row, i.e., number of `long` *objects that each row can contain.*

Friends

- class DB_Matrix

Related Functions

(Note that these are not member functions.)

- template<typename T >
  std::ostream & operator<< (std::ostream &s, const DB_Matrix< T > &c)

  Output operator.

- template<typename T >
  void swap (Parma_Polyhedra_Library::DB_Matrix< T > &x, Parma_Polyhedra_Library::DB_-Matrix< T > &y)

  Specializes `std::swap`.

- template<typename T >
  bool operator== (const DB_Matrix< T > &x, const DB_Matrix< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are identical.*

- template<typename T >
  bool operator!= (const DB_Matrix< T > &x, const DB_Matrix< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are different.*

- template<typename Temp , typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const DB_Matrix< T > &x, const DB_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`.

- template<typename Temp , typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const DB_Matrix< T > &x, const DB_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

Computes the euclidean distance between x and y.

- template<typename Temp , typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const DB_Matrix< T > &x, const DB_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ distance between x and y.

- template<typename Specialization , typename Temp , typename To , typename T >
  bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const DB_Matrix< T > &x, const DB_Matrix< T > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

11.42.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::DB_Matrix< T >

The base class for the square matrices. The template class DB_Matrix<T> allows for the representation of a square matrix of T objects. Each DB_Matrix<T> object can be viewed as a multiset of DB_Row<T>.

Definition at line 62 of file DB_Matrix.defs.hh.

11.42.2 Constructor & Destructor Documentation

11.42.2.1 template<typename T> Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix ()

Builds an empty matrix.

DB_Rows' size and capacity are initialized to 0.

11.42.2.2 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix (dimension_type *n_rows*) [inline, explicit]

Builds a square matrix having the specified dimension.

Definition at line 30 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::construct(), Parma_Polyhedra_Library::DB_Matrix< T >::OK(), Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, and Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.2.3 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix (const DB_Matrix< T > & *y*) [inline]

Copy constructor.

Definition at line 179 of file DB_Matrix.inlines.hh.

11.42.2.4 template<typename T > template<typename U > Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix (const DB_Matrix< U > & *y*) [inline, explicit]

Constructs a conservative approximation of `y`.

Definition at line 42 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::OK(), Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, and Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.2.5 template<typename T > Parma_Polyhedra_Library::DB_Matrix< T >::~DB_Matrix () [inline]

Destructor.

Definition at line 145 of file DB_Matrix.inlines.hh.

11.42.3 Member Function Documentation

11.42.3.1 template<typename T > void Parma_Polyhedra_Library::DB_Matrix< T >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 210 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::num_rows().

11.42.3.2 template<typename T> void Parma_Polyhedra_Library::DB_Matrix< T >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump().

11.42.3.3 template<typename T > bool Parma_Polyhedra_Library::DB_Matrix< T >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 228 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Checked::result_relation().

11.42.3.4 template<typename T > DB_Matrix< T >::const_iterator Parma_Polyhedra_Library::DB_Matrix< T >::begin () const [inline]

Returns the const_iterator pointing to the first row, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 125 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.3.5 template<typename T > DB_Matrix< T >::const_iterator Parma_Polyhedra_Library::DB_Matrix< T >::end () const [inline]

Returns the past-the-end const_iterator.

Definition at line 131 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.3.6 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Matrix< T >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 263 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::DB_-Matrix< T >::row_capacity, and Parma_Polyhedra_Library::DB_Matrix< T >::rows.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::external_memory_in_bytes(), and Parma_-Polyhedra_Library::DB_Matrix< T >::total_memory_in_bytes().

11.42.3.7 template<typename T > void Parma_Polyhedra_Library::DB_Matrix< T >::grow (dimension_type *new_n_rows*) [inline]

Makes the matrix grow by adding more rows and more columns.

Parameters

new_n_rows The number of rows and columns of the resized matrix.

A new matrix, with the specified dimension, is created. The contents of the old matrix are copied in the upper, left-hand corner of the new matrix, which is then assigned to `*this`.

Definition at line 54 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::construct(), Parma_Polyhedra_Library::DB_Matrix< T >::max_num_columns(), Parma_Polyhedra_Library::DB_-Matrix< T >::max_num_rows(), Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, Parma_-Polyhedra_Library::DB_Matrix< T >::row_size, Parma_Polyhedra_Library::DB_Matrix< T >::rows, and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::BD_Shape< T >::add_space_dimensions_and_project(), and Parma_Polyhedra_Library::BD_Shape< T >::concatenate_assign().

11.42.3.8 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Matrix< T >::max_num_columns () [inline, static]

Returns the maximum number of columns a DB_Matrix can handle.

Definition at line 51 of file DB_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::DB_Matrix< T >::operator=(), and Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy().

11.42.3.9 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Matrix< T >::max_num_rows () [inline, static]

Returns the maximum number of rows a DB_Matrix can handle.

Definition at line 45 of file DB_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::grow(), and Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy().

11.42.3.10 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Matrix< T >::num_rows () const [inline]

Returns the number of rows in the matrix.

Definition at line 164 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::rows.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::DB_Matrix< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::BD_Shape< T >::forget_all_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::forget_binary_dbm_constraints(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::DB_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::OK(), Parma_Polyhedra_Library::BD_Shape< T >::OK(), Parma_Polyhedra_Library::DB_Matrix< T >::operator==(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.42.3.11 template<typename T > bool Parma_Polyhedra_Library::DB_Matrix< T >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 272 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::DB_-Matrix< T >::row_capacity, and Parma_Polyhedra_Library::DB_Matrix< T >::row_size.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix(), and Parma_Polyhedra_-Library::BD_Shape< T >::OK().

11.42.3.12 template<typename T > DB_Matrix< T > & Parma_Polyhedra_-Library::DB_Matrix< T >::operator= (const DB_Matrix< T > & *y*) [inline]

Assignment operator.

Definition at line 187 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::DB_Matrix< T >::max_num_columns(), Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, Parma_-Polyhedra_Library::DB_Matrix< T >::row_size, and Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.3.13 template<typename T > const DB_Row< T > & Parma_Polyhedra_-Library::DB_Matrix< T >::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the `k-th` row of the matrix.

Definition at line 157 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.3.14 template<typename T > DB_Row< T > & Parma_Polyhedra_Library::DB_Matrix< T >::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the `k-th` row of the matrix.

Definition at line 150 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::rows.

11.42.3.15 template<typename T> void Parma_Polyhedra_Library::DB_Matrix< T >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.42.3.16 template<typename T > void Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy (dimension_type *new_n_rows*) [inline]

Resizes the matrix without worrying about the old contents.

Parameters

new_n_rows The number of rows and columns of the resized matrix.

A new matrix, with the specified dimension, is created without copying the content of the old matrix and assigned to `*this`.

Definition at line 137 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::construct(), Parma_Polyhedra_Library::DB_Matrix< T >::max_num_columns(), Parma_Polyhedra_Library::DB_Matrix< T >::max_num_rows(), Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, Parma_Polyhedra_Library::DB_Matrix< T >::row_size, Parma_Polyhedra_Library::DB_Matrix< T >::rows, and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::remove_higher_space_dimensions(), and Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions().

11.42.3.17 template<typename T > void Parma_Polyhedra_Library::DB_Matrix< T >::swap (DB_Matrix< T > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 37 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity, Parma_Polyhedra_Library::DB_Matrix< T >::row_size, and Parma_Polyhedra_Library::DB_Matrix< T >::rows.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

11.42.3.18 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Matrix< T >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 57 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::external_memory_in_bytes().

11.42.4 Friends And Related Function Documentation

11.42.4.1 template<typename T> Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix [friend]

Definition at line 161 of file DB_Matrix.defs.hh.

11.42.4.2 template<typename Temp , typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the Euclidean distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 283 of file DB_Matrix.inlines.hh.

11.42.4.3 template<typename Temp , typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 303 of file DB_Matrix.inlines.hh.

11.42.4.4 template<typename Specialization , typename Temp , typename To , typename T > bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Definition at line 208 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::finalize(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::maybe_assign(), Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::PLUS_INFINITY.

11.42.4.5 template<typename T > bool operator!= (const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 173 of file DB_Matrix.inlines.hh.

11.42.4.6 template<typename T > std::ostream & operator<< (std::ostream & *s*, const DB_Matrix< T > & *c*) [related]

Output operator.

Definition at line 305 of file DB_Matrix.templates.hh.

11.42.4.7 template<typename T > bool operator== (const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*) [related]

Returns `true` if and only if `x` and `y` are identical.

Definition at line 251 of file DB_Matrix.templates.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::num_rows().

11.42.4.8 template<typename Temp , typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const DB_Matrix< T > & *x*, const DB_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 262 of file DB_Matrix.inlines.hh.

11.42.4.9 template<typename T > void swap (Parma_Polyhedra_Library::DB_Matrix< T > & *x*, Parma_Polyhedra_Library::DB_Matrix< T > & *y*) [related]

Specializes `std::swap`.

Definition at line 327 of file DB_Matrix.inlines.hh.

References Parma_Polyhedra_Library::DB_Matrix< T >::swap().

11.42.5 Member Data Documentation

11.42.5.1 template<typename T> dimension_type Parma_Polyhedra_Library::DB_Matrix< T >::row_capacity [private]

Capacity allocated for each row, i.e., number of `long` objects that each row can contain.

Definition at line 173 of file DB_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::DB_Matrix(), Parma_Polyhedra_-Library::DB_Matrix< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::DB_Matrix< T >::OK(), Parma_Polyhedra_Library::DB_-Matrix< T >::operator=(), Parma_Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

11.42.5.2 template<typename T> dimension_type Parma_Polyhedra_Library::DB_Matrix< T >::row_size [private]

Size of the initialized part of each row.

Definition at line 167 of file DB_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::DB_-Matrix< T >::OK(), Parma_Polyhedra_Library::DB_Matrix< T >::operator=(), Parma_Polyhedra_-Library::DB_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

11.42.5.3 template<typename T> std::vector<DB_Row<T> > Parma_Polyhedra_-Library::DB_Matrix< T >::rows [private]

The rows of the matrix.

Definition at line 164 of file DB_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Matrix< T >::begin(), Parma_Polyhedra_Library::DB_-Matrix< T >::DB_Matrix(), Parma_Polyhedra_Library::DB_Matrix< T >::end(), Parma_Polyhedra_-Library::DB_Matrix< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Matrix< T >::grow(), Parma_Polyhedra_Library::DB_Matrix< T >::num_rows(), Parma_Polyhedra_Library::DB_-Matrix< T >::operator=(), Parma_Polyhedra_Library::DB_Matrix< T >::operator[ ](), Parma_-Polyhedra_Library::DB_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::DB_Matrix< T >::swap().

The documentation for this class was generated from the following files:

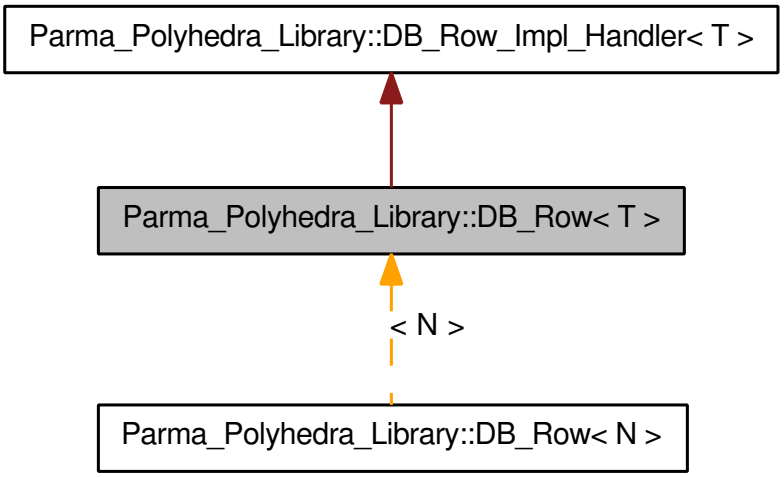
- DB_Matrix.defs.hh
- DB_Matrix.inlines.hh
- DB_Matrix.templates.hh

11.43 Parma_Polyhedra_Library::DB_Row< T > Class Template Reference

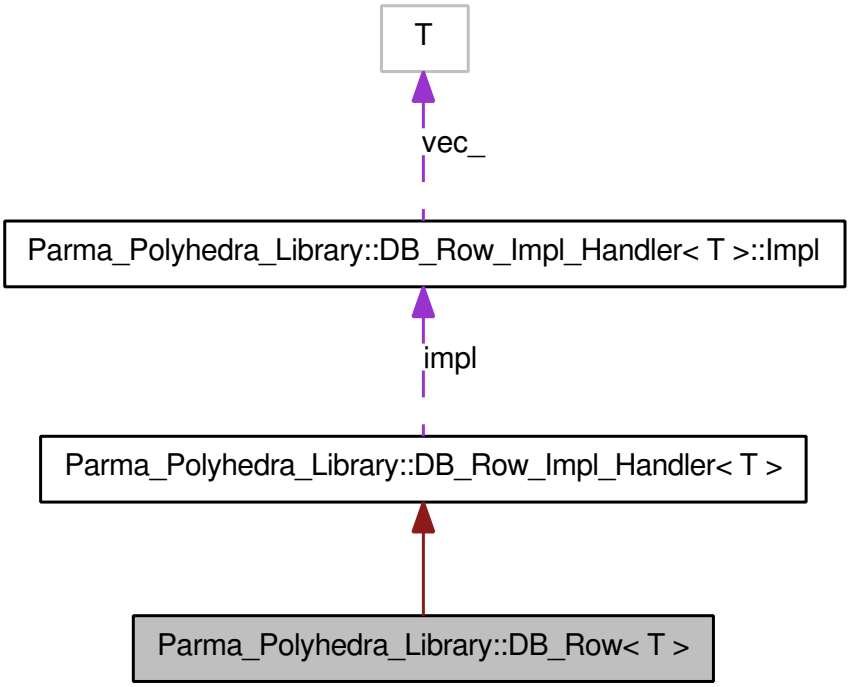
The base class for the single rows of matrices.

`#include <DB_Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::DB_Row< T >:



Collaboration diagram for Parma_Polyhedra_Library::DB_Row< T >:



Public Types

- typedef Implementation::Ptr_Iterator< T * > iterator

  A (non const) random access iterator to access the row's elements.

- typedef Implementation::Ptr_Iterator< const T * > const_iterator

  A const random access iterator to access the row's elements.

Public Member Functions

- DB_Row ()

  Pre-constructs a row: construction must be completed by construct().

- DB_Row (dimension_type sz)

  Tight constructor: resizing will require reallocation.

- DB_Row (dimension_type sz, dimension_type capacity)

  Sizing constructor with capacity.

- DB_Row (const DB_Row &y)

  Ordinary copy constructor.

- DB_Row (const DB_Row &y, dimension_type capacity)

  Copy constructor with specified capacity.

- DB_Row (const DB_Row &y, dimension_type sz, dimension_type capacity)

  Copy constructor with specified size and capacity.

- ~DB_Row ()

  Destructor.

- DB_Row & operator= (const DB_Row &y)

  Assignment operator.

- void swap (DB_Row &y)

  Swaps `*this` *with* `y`.

- void assign (DB_Row &y)

  Assigns the implementation of `y` *to* `*this`.

- void allocate (dimension_type capacity)

  Allocates memory for a default constructed DB_Row object, allowing for `capacity` *coefficients at most.*

- void expand_within_capacity (dimension_type new_size)

  Expands the row to size `new_size`.

- void shrink (dimension_type new_size)

  Shrinks the row by erasing elements at the end.

- dimension_type size () const

  Gives the number of coefficients currently in use.

- iterator begin ()

  Returns the const iterator pointing to the first element, if `*this` *is not empty; otherwise, returns the past-the-end const iterator.*

- iterator end ()

  Returns the past-the-end iterator.

- const_iterator begin () const

  Returns the const iterator pointing to the first element, if `*this` *is not empty; otherwise, returns the past-the-end const iterator.*

- const_iterator end () const

  Returns the past-the-end const iterator.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns a lower bound to the size in bytes of the memory managed by `*this`.

- memory_size_type total_memory_in_bytes (dimension_type capacity) const

  Returns the total size in bytes of the memory occupied by `*this`*, provided the capacity of* `*this` *is given by* `capacity`.

- memory_size_type external_memory_in_bytes (dimension_type capacity) const

  Returns the size in bytes of the memory managed by `*this`*, provided the capacity of* `*this` *is given by* `capacity`.

- bool OK (dimension_type row_size, dimension_type row_capacity) const

  Checks if all the invariants are satisfied.

Post-constructors.

- void construct (dimension_type sz)

  Constructs properly a default-constructed element.

- void construct (dimension_type sz, dimension_type capacity)

  Constructs properly a default-constructed element.

- template<typename U >
  void construct_upward_approximation (const DB_Row< U > &y, dimension_type capacity)

  Constructs properly a conservative approximation of `y`.

Subscript operators.

- T & operator[ ] (dimension_type k)

  Returns a reference to the element of the row indexed by `k`.

- const T & operator[ ] (dimension_type k) const

  Returns a constant reference to the element of the row indexed by `k`.

Static Public Member Functions

- static dimension_type max_size ()

  Returns the size() of the largest possible DB_Row.

Private Member Functions

- void copy_construct_coefficients (const DB_Row &y)

  Exception-safe copy construction mechanism for coefficients.

Friends

- class Parma_Polyhedra_Library::DB_Row

Related Functions

(Note that these are not member functions.)

- template<typename T >
  void swap (Parma_Polyhedra_Library::DB_Row< T > &x, Parma_Polyhedra_Library::DB_Row< T > &y)

  Specializes `std::swap`.

- template<typename T >
  void iter_swap (typename std::vector< Parma_Polyhedra_Library::DB_Row< T > >::iterator x, typename std::vector< Parma_Polyhedra_Library::DB_Row< T > >::iterator y)

  Specializes `std::iter_swap`.

Classical comparison operators.

- template<typename T >
  bool operator== (const DB_Row< T > &x, const DB_Row< T > &y)
- template<typename T >
  bool operator!= (const DB_Row< T > &x, const DB_Row< T > &y)

11.43.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::DB_Row< T >

The base class for the single rows of matrices. The class template DB_Row<T> allows for the efficient representation of the single rows of a DB_Matrix. It contains elements of type T stored as a vector. The class T is a family of extended numbers that must provide representation for $-\infty$, 0,$+\infty$ (and, consequently for *nan*, *not a number*, since this arises as the "result" of undefined sums like $+\infty + (-\infty)$).

The class T must provide the following methods:

```
T()
```

is the default constructor: no assumption is made on the particular object constructed, provided `T().OK()` gives `true` (see below).

```
~T()
```

is the destructor.

```
bool is_nan() const
```

returns `true` if and only `*this` represents the *not a number* value.

```
bool OK() const
```

returns `true` if and only if `*this` satisfies all its invariants.

Definition at line 120 of file DB_Row.defs.hh.

11.43.2 Member Typedef Documentation

11.43.2.1 template<typename T> typedef Implementation::Ptr_Iterator<const T*> Parma_Polyhedra_Library::DB_Row< T >::const_iterator

A const random access iterator to access the row's elements.

Definition at line 247 of file DB_Row.defs.hh.

11.43.2.2 template<typename T> typedef Implementation::Ptr_Iterator<T*> Parma_Polyhedra_Library::DB_Row< T >::iterator

A (non const) random access iterator to access the row's elements.

Definition at line 244 of file DB_Row.defs.hh.

11.43.3 Constructor & Destructor Documentation

11.43.3.1 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row () [inline]

Pre-constructs a row: construction must be completed by construct().

Definition at line 168 of file DB_Row.inlines.hh.

11.43.3.2 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row (dimension_type *sz*) [inline]

Tight constructor: resizing will require reallocation.

Definition at line 252 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::construct().

11.43.3.3 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row (dimension_type *sz*, dimension_type *capacity*) [inline]

Sizing constructor with capacity.

Definition at line 244 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::construct().

11.43.3.4 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row (const DB_Row< T > & *y*) [inline]

Ordinary copy constructor.

Definition at line 258 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row< T >::max_size(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.3.5 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row (const DB_Row< T > & *y*, dimension_type *capacity*) [inline]

Copy constructor with specified capacity.

It is assumed that `capacity` is greater than or equal to `y` size.

Definition at line 268 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row< T >::max_size(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.3.6 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::DB_Row (const DB_Row< T > & *y*, dimension_type *sz*, dimension_type *capacity*) [inline]

Copy constructor with specified size and capacity.

It is assumed that `sz` is greater than or equal to the size of `y` and, of course, that `sz` is less than or equal to `capacity`. Any new position is initialized to $+\infty$.

Definition at line 279 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row< T >::expand_within_capacity(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row< T >::max_size(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.3.7 template<typename T > Parma_Polyhedra_Library::DB_Row< T >::~DB_Row () [inline]

Destructor.

Definition at line 292 of file DB_Row.inlines.hh.

11.43.4 Member Function Documentation

11.43.4.1 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::allocate (dimension_type *capacity*) [inline]

Allocates memory for a default constructed DB_Row object, allowing for `capacity` coefficients at most.

It is assumed that no allocation has been performed before (otherwise, a memory leak will occur). After execution, the size of the DB_Row object is zero.

Definition at line 174 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row< T >::max_size().

Referenced by Parma_Polyhedra_Library::DB_Row< T >::construct(), Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation(), and Parma_Polyhedra_Library::DB_Row< T >::DB_Row().

11.43.4.2 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::assign (DB_Row< T > & *y*) [inline]

Assigns the implementation of `y` to `*this`.

Definition at line 315 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

11.43.4.3 template<typename T > DB_Row< T >::const_iterator Parma_Polyhedra_Library::DB_Row< T >::begin () const [inline]

Returns the const iterator pointing to the first element, if `*this` is not empty; otherwise, returns the past-the-end const iterator.

Definition at line 364 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.43.4.4 template<typename T > DB_Row< T >::iterator Parma_Polyhedra_Library::DB_Row< T >::begin () [inline]

Returns the const iterator pointing to the first element, if `*this` is not empty; otherwise, returns the past-the-end const iterator.

Definition at line 350 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.43.4.5 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::construct (dimension_type *sz*, dimension_type *capacity*) [inline]

Constructs properly a default-constructed element.

Parameters

sz The size of the row that will be constructed.

capacity The minimum capacity of the row that will be constructed.

The row that we are constructing has a minimum capacity of (i.e., it can contain at least) `elements`, `sz` of which will be constructed now.

Definition at line 229 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row< T >::expand_within_capacity(), and Parma_Polyhedra_Library::DB_Row< T >::max_size().

11.43.4.6 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::construct (dimension_type *sz*) [inline]

Constructs properly a default-constructed element.

Builds a row with size `sz` and minimum capacity.

Definition at line 238 of file DB_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::DB_Row().

11.43.4.7 template<typename T > template<typename U > void Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation (const DB_Row< U > & *y*, dimension_type *capacity*) [inline]

Constructs properly a conservative approximation of `y`.

Parameters

y A row containing the elements whose upward approximations will be used to properly construct `*this`.

capacity The capacity of the constructed row.

It is assumed that `capacity` is greater than or equal to the size of `y`.

Definition at line 218 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::construct_upward_approximation(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row< T >::max_size(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.4.8 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients (const DB_Row< T > & *y*) [inline, private]

Exception-safe copy construction mechanism for coefficients.

Definition at line 206 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row< T >::size().

Referenced by Parma_Polyhedra_Library::DB_Row< T >::DB_Row().

11.43.4.9 template<typename T > DB_Row< T >::const_iterator Parma_Polyhedra_Library::DB_Row< T >::end () const [inline]

Returns the past-the-end const iterator.

Definition at line 371 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.43.4.10 template<typename T > DB_Row< T >::iterator Parma_Polyhedra_Library::DB_Row< T >::end () [inline]

Returns the past-the-end iterator.

Definition at line 357 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.43.4.11 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::expand_within_capacity (dimension_type *new_size*) [inline]

Expands the row to size `new_size`.

Adds new positions to the implementation of the row obtaining a new row with size `new_size`. It is assumed that `new_size` is between the current size and capacity of the row. The new positions are initialized to $+\infty$.

Definition at line 195 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::construct(), and Parma_Polyhedra_Library::DB_Row< T >::DB_Row().

11.43.4.12 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the size in bytes of the memory managed by `*this`, provided the capacity of `*this` is given by `capacity`.

Definition at line 378 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes().

11.43.4.13 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by `*this`.

Definition at line 391 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes().

Referenced by Parma_Polyhedra_Library::DB_Row< T >::total_memory_in_bytes().

11.43.4.14 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Row< T >::max_size () [inline, static]

Returns the size() of the largest possible DB_Row.

Definition at line 148 of file DB_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row< T >::construct(), Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation(), and Parma_Polyhedra_Library::DB_Row< T >::DB_Row().

11.43.4.15 template<typename T > bool Parma_Polyhedra_Library::DB_Row< T >::OK (dimension_type *row_size*, dimension_type *row_capacity*) const [inline]

Checks if all the invariants are satisfied.

Definition at line 126 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::is_not_a_number(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.4.16 template<typename T > DB_Row< T > & Parma_Polyhedra_Library::DB_Row< T >::operator= (const DB_Row< T > & *y*) [inline]

Assignment operator.

Definition at line 325 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::swap().

11.43.4.17 template<typename T > const T & Parma_Polyhedra_Library::DB_Row< T >::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the element of the row indexed by `k`.

Definition at line 343 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

11.43.4.18 template<typename T > T & Parma_Polyhedra_Library::DB_Row< T >::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the element of the row indexed by `k`.

Definition at line 336 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

11.43.4.19 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::shrink (dimension_type *new_size*) [inline]

Shrinks the row by erasing elements at the end.

Destroys elements of the row implementation from position `new_size` to the end. It is assumed that `new_size` is not greater than the current size.

Definition at line 297 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_-Library::DB_Row_Impl_Handler< T >::Impl::shrink().

11.43.4.20 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Row< T >::size () const [inline]

Gives the number of coefficients currently in use.

Definition at line 154 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl, and Parma_Polyhedra_-Library::DB_Row_Impl_Handler< T >::Impl::size().

Referenced by Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation(), Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row< T >::DB_Row(), Parma_Polyhedra_Library::DB_Row< T >::OK(), and Parma_Polyhedra_Library::DB_Row< T >::operator==().

11.43.4.21 template<typename T > void Parma_Polyhedra_Library::DB_Row< T >::swap (DB_Row< T > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 305 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::operator=(), and Parma_Polyhedra_Library::DB_Row< T >::swap().

11.43.4.22 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row< T >::total_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the total size in bytes of the memory occupied by `*this`, provided the capacity of `*this` is given by `capacity`.

Definition at line 385 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes().

11.43.4.23 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row< T >::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Definition at line 402 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes().

11.43.5 Friends And Related Function Documentation

11.43.5.1 template<typename T > void iter_swap (typename std::vector< Parma_Polyhedra_Library::DB_Row< T > >::iterator *x*, typename std::vector< Parma_Polyhedra_Library::DB_Row< T > >::iterator *y*) [related]

Specializes `std::iter_swap`.

Definition at line 429 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::swap().

11.43.5.2 template<typename T > bool operator!= (const DB_Row< T > & *x*, const DB_Row< T > & *y*) [related]

Definition at line 409 of file DB_Row.inlines.hh.

11.43.5.3 template<typename T > bool operator== (const DB_Row< T > & *x*, const DB_Row< T > & *y*) [related]

Definition at line 204 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::DB_Row< T >::size().

11.43.5.4 template<typename T> friend class Parma_Polyhedra_Library::DB_Row [friend]

Definition at line 297 of file DB_Row.defs.hh.

11.43.5.5 template<typename T > void swap (Parma_Polyhedra_Library::DB_Row< T > & *x*, Parma_Polyhedra_Library::DB_Row< T > & *y*) [related]

Specializes `std::swap`.

Definition at line 421 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row< T >::swap().

The documentation for this class was generated from the following files:

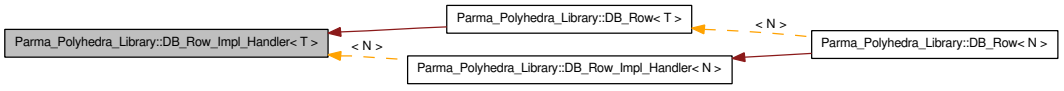
- DB_Row.defs.hh
- DB_Row.inlines.hh
- DB_Row.templates.hh

11.44 Parma_Polyhedra_Library::DB_Row_Impl_Handler< T > Class Template Reference

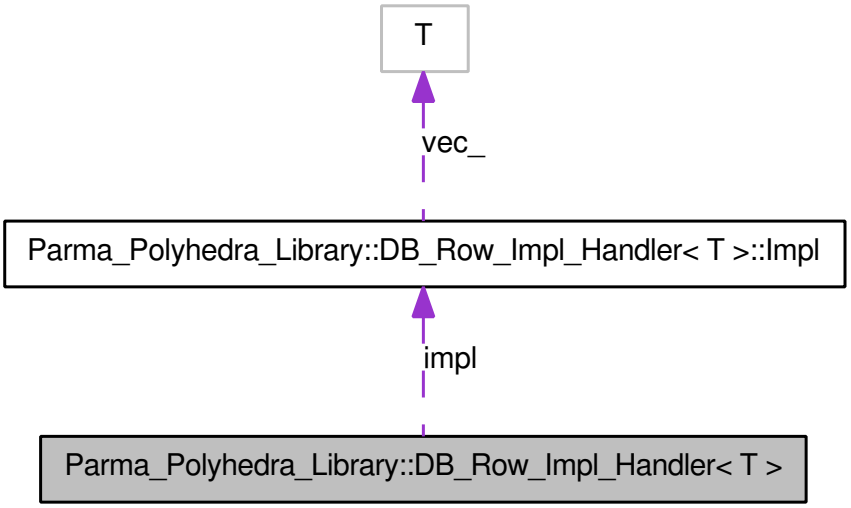
The handler of the actual DB_Row implementation.

`#include <DB_Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >:



Collaboration diagram for Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >:



Classes

- class Impl

  The real implementation of a DB_Row object.

Public Member Functions

- DB_Row_Impl_Handler ()

  Default constructor.

- ~DB_Row_Impl_Handler ()

  Destructor.

Public Attributes

- Impl * impl

  A pointer to the actual implementation.

Private Member Functions

- DB_Row_Impl_Handler (const DB_Row_Impl_Handler &)

  Private and unimplemented: copy construction is not allowed.

- DB_Row_Impl_Handler & operator= (const DB_Row_Impl_Handler &)

  Private and unimplemented: copy assignment is not allowed.

11.44.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >

The handler of the actual DB_Row implementation. Exception-safety is the only responsibility of this class: it has to ensure that its `impl` member is correctly deallocated.

Definition at line 58 of file DB_Row.defs.hh.

11.44.2 Constructor & Destructor Documentation

11.44.2.1 template<typename T > Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::DB_Row_Impl_Handler () [inline]

Default constructor.

Definition at line 119 of file DB_Row.inlines.hh.

11.44.2.2 template<typename T > Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::~DB_Row_Impl_Handler () [inline]

Destructor.

Definition at line 128 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl.

11.44.2.3 template<typename T> Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::DB_Row_Impl_Handler (const DB_Row_Impl_Handler< T > &) [private]

Private and unimplemented: copy construction is not allowed.

11.44.3 Member Function Documentation

11.44.3.1 template<typename T> DB_Row_Impl_Handler& Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::operator= (const DB_Row_Impl_Handler< T > &) [private]

Private and unimplemented: copy assignment is not allowed.

11.44.4 Member Data Documentation

11.44.4.1 template<typename T> Impl* Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::impl

A pointer to the actual implementation.

Definition at line 66 of file DB_Row.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::allocate(), Parma_Polyhedra_Library::DB_Row< T >::assign(), Parma_Polyhedra_Library::DB_Row< T >::begin(), Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation(), Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row< T >::DB_Row(), Parma_Polyhedra_Library::DB_Row< T >::end(), Parma_Polyhedra_Library::DB_Row< T >::expand_within_capacity(), Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Row< T >::operator[ ](), Parma_Polyhedra_Library::DB_Row< T >::shrink(), Parma_Polyhedra_Library::DB_Row< T >::size(), Parma_Polyhedra_Library::DB_Row< T >::swap(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::~DB_Row_Impl_Handler().

The documentation for this class was generated from the following files:

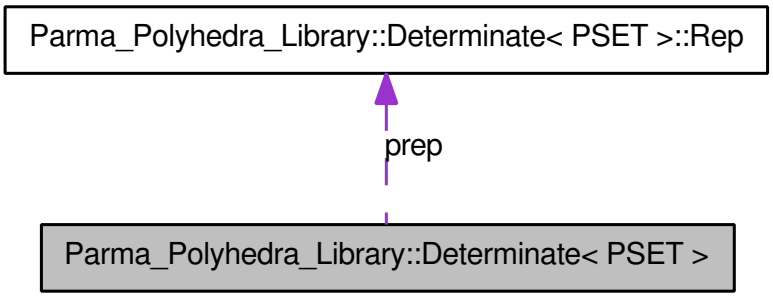
- DB_Row.defs.hh
- DB_Row.inlines.hh

11.45 Parma_Polyhedra_Library::Determinate< PSET > Class Template Reference

A wrapper for PPL pointsets, providing them with a *determinate constraint system* interface, as defined in [Bag98].

`#include <Determinate.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Determinate< PSET >:



Classes

- class Binary_Operator_Assign_Lifter

  A function adapter for the Determinate class.

- class Rep

  The possibly shared representation of a Determinate object.

Public Member Functions

Constructors and Destructor

- Determinate (const PSET &p)
  Constructs a COW-wrapped object corresponding to the pointset `p`.
- Determinate (const Constraint_System &cs)
  Constructs a COW-wrapped object corresponding to the pointset defined by `cs`.
- Determinate (const Congruence_System &cgs)
  Constructs a COW-wrapped object corresponding to the pointset defined by `cgs`.
- Determinate (const Determinate &y)
  Copy constructor.
- ~Determinate ()
  Destructor.

Member Functions that May Modify the Domain Element

- void upper_bound_assign (const Determinate &y)
  Assigns to `*this` *the upper bound of* `*this` *and* `y`.
- void meet_assign (const Determinate &y)
  Assigns to `*this` *the meet of* `*this` *and* `y`.
- void weakening_assign (const Determinate &y)
  Assigns to `*this` *the result of weakening* `*this` *with* `y`.
- void concatenate_assign (const Determinate &y)
  Assigns to `*this` *the concatenation of* `*this` *and* `y`*, taken in this order.*
- PSET & pointset ()
  Returns a reference to the embedded element.
- void mutate ()
  On return from this method, the representation of `*this` *is not shared by different Determinate objects.*
- Determinate & operator= (const Determinate &y)
  Assignment operator.
- void swap (Determinate &y)
  Swaps `*this` *with* `y`.

Static Public Member Functions

- template<typename Binary_Operator_Assign >
  static Binary_Operator_Assign_Lifter< Binary_Operator_Assign > lift_op_assign (Binary_Operator_Assign op_assign)

  Helper function returning a Binary_Operator_Assign_Lifter object, also allowing for the deduction of template arguments.

Private Attributes

- Rep * prep

  A pointer to the possibly shared representation of the base-level domain element.

Friends

- bool operator== (const Determinate< PSET > &x, const Determinate< PSET > &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same COW-wrapped pointset.*

- bool operator!= (const Determinate< PSET > &x, const Determinate< PSET > &y)

  Returns `true` *if and only if* `x` *and* `y` *are different COW-wrapped pointsets.*

Related Functions

(Note that these are not member functions.)

- template<typename PSET >
  std::ostream & operator<< (std::ostream &, const Determinate< PSET > &)

  Output operator.

- template<typename PSET >
  void swap (Parma_Polyhedra_Library::Determinate< PSET > &x, Parma_Polyhedra_Library::Determinate< PSET > &y)

  Specializes `std::swap`.

Member Functions that Do Not Modify the Domain Element

- const PSET & pointset () const

  Returns a const reference to the embedded pointset.

- bool is_top () const

  Returns `true` *if and only if* `*this` *embeds the universe element* `PSET`.

- bool is_bottom () const

  Returns `true` *if and only if* `*this` *embeds the empty element of* `PSET`.

- bool definitely_entails (const Determinate &y) const

  Returns `true` *if and only if* `*this` *entails* `y`.

- bool is_definitely_equivalent_to (const Determinate &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are definitely equivalent.*

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns a lower bound to the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

- static bool has_nontrivial_weakening ()

11.45.1 Detailed Description

template<typename PSET> class Parma_Polyhedra_Library::Determinate< PSET >

A wrapper for PPL pointsets, providing them with a *determinate constraint system* interface, as defined in [Bag98]. The implementation uses a copy-on-write optimization, making the class suitable for constructions, like the *finite powerset* and *ask-and-tell* of [Bag98], that are likely to perform many copies.

Definition at line 80 of file Determinate.defs.hh.

11.45.2 Constructor & Destructor Documentation

11.45.2.1 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Determinate (const PSET & *p*) [inline]

Constructs a COW-wrapped object corresponding to the pointset `p`.

Definition at line 94 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.2.2 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Determinate (const Constraint_System & *cs*) [inline]

Constructs a COW-wrapped object corresponding to the pointset defined by `cs`.

Definition at line 101 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.2.3 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Determinate (const Congruence_System & *cgs*) [inline]

Constructs a COW-wrapped object corresponding to the pointset defined by `cgs`.

Definition at line 108 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.2.4 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Determinate (const Determinate< PSET > & *y*) [inline]

Copy constructor.

Definition at line 115 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.2.5 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::~Determinate () [inline]

Destructor.

Definition at line 122 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::del_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.3 Member Function Documentation

11.45.3.1 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::concatenate_assign (const Determinate< PSET > & *y*) [inline]

Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

Definition at line 199 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::pointset().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign().

11.45.3.2 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::definitely_entails (const Determinate< PSET > & *y*) const [inline]

Returns `true` if and only if `*this` entails `y`.

Definition at line 205 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

11.45.3.3 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Determinate< PSET >::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by `*this`.

Definition at line 229 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::total_memory_in_bytes().

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::total_memory_in_bytes().

11.45.3.4 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::has_nontrivial_weakening () [inline, static]

Returns `true` if and only if this domain has a nontrivial weakening operator.

Definition at line 181 of file Determinate.inlines.hh.

11.45.3.5 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::is_bottom () const [inline]

Returns `true` if and only if `*this` embeds the empty element of `PSET`.

Definition at line 223 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign().

11.45.3.6 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::is_definitely_equivalent_to (const Determinate< PSET > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are definitely equivalent.

Definition at line 211 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

11.45.3.7 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::is_top () const [inline]

Returns `true` if and only if `*this` embeds the universe element `PSET`.

Definition at line 217 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

11.45.3.8 template<typename PSET > template<typename Binary_Operator_Assign > Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign > Parma_Polyhedra_Library::Determinate< PSET >::lift_op_assign (Binary_Operator_Assign *op_assign*) [inline, static]

Helper function returning a Binary_Operator_Assign_Lifter object, also allowing for the deduction of template arguments.

Definition at line 291 of file Determinate.inlines.hh.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::time_elapse_assign().

11.45.3.9 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::meet_assign (const Determinate< PSET > & *y*) [inline]

Assigns to `*this` the meet of `*this` and `y`.

Definition at line 175 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.45.3.10 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::mutate () [inline]

On return from this method, the representation of `*this` is not shared by different Determinate objects.

Definition at line 145 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::del_reference(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::is_shared(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.45.3.11 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 241 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

11.45.3.12 template<typename PSET > Determinate< PSET > & Parma_Polyhedra_Library::Determinate< PSET >::operator= (const Determinate< PSET > & *y*) [inline]

Assignment operator.

Definition at line 129 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::del_reference(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::prep.

11.45.3.13 template<typename PSET > PSET & Parma_Polyhedra_Library::Determinate< PSET >::pointset () [inline]

Returns a reference to the embedded element.

Definition at line 162 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::mutate(), Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

11.45.3.14 template<typename PSET > const PSET & Parma_Polyhedra_Library::Determinate< PSET >::pointset () const [inline]

Returns a const reference to the embedded pointset.

Definition at line 156 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep, and Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Determinate< PSET >::meet_assign(), Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >::operator()(), Parma_Polyhedra_Library::Determinate< PSET >::upper_bound_assign(), and Parma_Polyhedra_Library::Determinate< PSET >::weakening_assign().

11.45.3.15 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::swap (Determinate< PSET > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 139 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::prep.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::swap().

11.45.3.16 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Determinate< PSET >::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Definition at line 235 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::external_memory_in_bytes().

11.45.3.17 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::upper_bound_assign (const Determinate< PSET > & *y*) [inline]

Assigns to `*this` the upper bound of `*this` and `y`.

Definition at line 169 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.45.3.18 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::weakening_assign (const Determinate< PSET > & *y*) [inline]

Assigns to `*this` the result of weakening `*this` with `y`.

Definition at line 190 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.45.4 Friends And Related Function Documentation

11.45.4.1 template<typename PSET> bool operator!= (const Determinate< PSET > & *x*, const Determinate< PSET > & *y*) [friend]

Returns `true` if and only if `x` and `y` are different COW-wrapped pointsets.

Definition at line 267 of file Determinate.inlines.hh.

11.45.4.2 template<typename PSET > std::ostream & operator<< (std::ostream & *s*, const Determinate< PSET > & *x*) [related]

Output operator.

Definition at line 250 of file Determinate.inlines.hh.

11.45.4.3 template<typename PSET> bool operator== (const Determinate< PSET > & *x*, const Determinate< PSET > & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same COW-wrapped pointset.

Definition at line 260 of file Determinate.inlines.hh.

11.45.4.4 template<typename PSET > void swap (Parma_Polyhedra_Library::Determinate< PSET > & *x*, Parma_Polyhedra_Library::Determinate< PSET > & *y*) [related]

Specializes `std::swap`.

Definition at line 303 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::swap().

11.45.5 Member Data Documentation

11.45.5.1 template<typename PSET> Rep* Parma_Polyhedra_Library::Determinate< PSET >::prep [private]

A pointer to the possibly shared representation of the base-level domain element.

Definition at line 315 of file Determinate.defs.hh.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::definitely_entails(), Parma_Polyhedra_Library::Determinate< PSET >::Determinate(), Parma_Polyhedra_Library::Determinate< PSET >::external_memory_in_bytes(), Parma_Polyhedra_Library::Determinate< PSET >::is_bottom(), Parma_Polyhedra_Library::Determinate< PSET >::is_definitely_equivalent_to(), Parma_Polyhedra_Library::Determinate< PSET >::is_top(), Parma_Polyhedra_Library::Determinate< PSET >::mutate(), Parma_Polyhedra_Library::Determinate< PSET >::OK(), Parma_Polyhedra_Library::Determinate< PSET >::operator=(), Parma_Polyhedra_Library::Determinate< PSET >::pointset(), Parma_Polyhedra_Library::Determinate< PSET >::swap(), and Parma_Polyhedra_Library::Determinate< PSET >::~Determinate().

The documentation for this class was generated from the following files:

- Determinate.defs.hh
- Determinate.inlines.hh

11.46 Parma_Polyhedra_Library::Domain_Product< D1, D2 > Class Template Reference

This class is temporary and will be removed when template typedefs will be supported in C++.

`#include <Partially_Reduced_Product.defs.hh>`

Public Types

- typedef Partially_Reduced_Product< D1, D2, No_Reduction< D1, D2 > > Direct_Product

- typedef Partially_Reduced_Product< D1, D2, Smash_Reduction< D1, D2 > > Smash_Product
- typedef Partially_Reduced_Product< D1, D2, Constraints_Reduction< D1, D2 > > Constraints_-Product
- typedef Partially_Reduced_Product< D1, D2, Congruences_Reduction< D1, D2 > > Congruences_Product
- typedef Partially_Reduced_Product< D1, D2, Shape_Preserving_Reduction< D1, D2 > > Shape_-Preserving_Product

11.46.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::Domain_Product< D1, D2 >

This class is temporary and will be removed when template typedefs will be supported in C++. When template typedefs will be supported in C++, what now is verbosely denoted by Domain_Product<Domain1, Domain2>::Direct_Product will simply be denoted by Direct_Product<Domain1, Domain2>.

Definition at line 1656 of file Partially_Reduced_Product.defs.hh.

11.46.2 Member Typedef Documentation

11.46.2.1 template<typename D1 , typename D2 > typedef Partially_Reduced_Product<D1, D2, Congruences_Reduction<D1, D2> > Parma_Polyhedra_Library::Domain_Product< D1, D2 >::Congruences_Product

Definition at line 1668 of file Partially_Reduced_Product.defs.hh.

11.46.2.2 template<typename D1 , typename D2 > typedef Partially_Reduced_Product<D1, D2, Constraints_Reduction<D1, D2> > Parma_Polyhedra_Library::Domain_Product< D1, D2 >::Constraints_Product

Definition at line 1665 of file Partially_Reduced_Product.defs.hh.

11.46.2.3 template<typename D1 , typename D2 > typedef Partially_Reduced_Product<D1, D2, No_Reduction<D1, D2> > Parma_Polyhedra_Library::Domain_Product< D1, D2 >::Direct_Product

Definition at line 1659 of file Partially_Reduced_Product.defs.hh.

11.46.2.4 template<typename D1 , typename D2 > typedef Partially_-Reduced_Product<D1, D2, Shape_Preserving_Reduction<D1, D2> > Parma_Polyhedra_Library::Domain_Product< D1, D2 >::Shape_Preserving_Product

Definition at line 1671 of file Partially_Reduced_Product.defs.hh.

11.46.2.5 template<typename D1 , typename D2 > typedef Partially_Reduced_Product<D1, D2, Smash_Reduction<D1, D2> > Parma_Polyhedra_Library::Domain_Product< D1, D2 >::Smash_Product

Definition at line 1662 of file Partially_Reduced_Product.defs.hh.

The documentation for this class was generated from the following file:

- Partially_Reduced_Product.defs.hh

11.47 Parma_Polyhedra_Library::Enable_If< b, T > Struct Template Reference

A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`.

`#include <meta_programming.hh>`

11.47.1 Detailed Description

template<bool b, typename T = void> struct Parma_Polyhedra_Library::Enable_If< b, T >

A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`. This is the non-specialized case, in which the `type` member is not present.

Definition at line 271 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.48 Parma_Polyhedra_Library::Enable_If< true, T > Struct Template Reference

A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`.

`#include <meta_programming.hh>`

Public Types

- typedef T type

11.48.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Enable_If< true, T >

A class that provides a type member called `type` equivalent to `T` if and only if `b` is `true`. This is the specialization in which the `type` member is present.

Note

Let `T`, `T1` and `T2` be any type expressions and suppose we have some template function `T f(T1, T2)`. If we want to declare a specialization that is enabled only if some compile-time checkable condition holds, we simply declare the specialization by

```
template ...
typename Enable_If<condition, T>::type
foo(T1 x, T2 y);
```

For all the instantiations of the template parameters that cause `condition` to evaluate to `false`, the `Enable_If<condition, T>::type` member will not be defined. Hence, for that instantiations, the specialization will not be eligible.

Definition at line 306 of file meta_programming.hh.

11.48.2 Member Typedef Documentation

11.48.2.1 template<typename T > typedef T Parma_Polyhedra_Library::Enable_If< true, T >::type

Definition at line 307 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.49 Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T > Struct Template Reference

`#include <meta_programming.hh>`

Public Types

- typedef T type

11.49.1 Detailed Description

template<typename Type, Type, typename T = void> struct Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T >

Definition at line 275 of file meta_programming.hh.

11.49.2 Member Typedef Documentation

11.49.2.1 template<typename Type , Type , typename T = void> typedef T Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T >::type

Definition at line 276 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.50 Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp > Struct Template Reference

`#include <distances.defs.hh>`

Static Public Member Functions

- static void combine (Temp &running, Temp &current, Rounding_Dir dir)
- static void finalize (Temp &running, Rounding_Dir dir)

11.50.1 Detailed Description

template<typename Temp> struct Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp >

Definition at line 38 of file distances.defs.hh.

11.50.2 Member Function Documentation

11.50.2.1 template<typename Temp > void Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp >::combine (Temp & *running*, Temp & *current*, Rounding_Dir *dir*) [inline, static]

Definition at line 83 of file distances.inlines.hh.

11.50.2.2 template<typename Temp > void Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp >::finalize (Temp & *running*, Rounding_Dir *dir*) [inline, static]

Definition at line 92 of file distances.inlines.hh.

The documentation for this struct was generated from the following files:

- distances.defs.hh
- distances.inlines.hh

11.51 Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type > Struct Template Reference

`#include <checked_int.inlines.hh>`

Static Public Attributes

- static const Type plus_infinity = C_Integer<Type>::max
- static const Type minus_infinity
- static const Type not_a_number

- static const Type min
- static const Type max

11.51.1 Detailed Description

template<typename Policy, typename Type> struct Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >

Definition at line 72 of file checked_int.inlines.hh.

11.51.2 Member Data Documentation

11.51.2.1 template<typename Policy , typename Type > const Type Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >::max [static]

Initial value:

```
(C_Integer<Type>::max
                          - (C_Integer<Type>::min >= 0
                             ? (2 * Policy::has_infinity + Policy::has_nan)
                             : Policy::has_infinity))
```

Definition at line 83 of file checked_int.inlines.hh.

11.51.2.2 template<typename Policy , typename Type > const Type Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >::min [static]

Initial value:

```
(C_Integer<Type>::min
                         + (C_Integer<Type>::min >= 0 ? 0
                            : (Policy::has_infinity + Policy::has_nan)))
```

Definition at line 80 of file checked_int.inlines.hh.

11.51.2.3 template<typename Policy , typename Type > const Type Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >::minus_infinity [static]

Initial value:

```
(C_Integer<Type>::min >= 0
                                     ? C_Integer<Type>::max - 1
                                     : C_Integer<Type>::min)
```

Definition at line 74 of file checked_int.inlines.hh.

11.51.2.4 template<typename Policy , typename Type > const Type Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >::not_a_number [static]

Initial value:

```
(C_Integer<Type>::min >= 0
                                    ? C_Integer<Type>::max - Policy::has_infinity
      * 2
                                    : C_Integer<Type>::min + Policy::has_infinity
     )
```

Definition at line 77 of file checked_int.inlines.hh.

11.51.2.5 template<typename Policy , typename Type > const Type Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >::plus_infinity = C_Integer<Type>::max [static]

Definition at line 73 of file checked_int.inlines.hh.

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.52 Parma_Polyhedra_Library::Extended_Number_Policy Struct Reference

`#include <Checked_Number.defs.hh>`

Public Member Functions

- const_bool_nodef (check_overflow, true)
- const_bool_nodef (check_inf_add_inf, false)
- const_bool_nodef (check_inf_sub_inf, false)
- const_bool_nodef (check_inf_mul_zero, false)
- const_bool_nodef (check_div_zero, false)
- const_bool_nodef (check_inf_div_inf, false)
- const_bool_nodef (check_inf_mod, false)
- const_bool_nodef (check_sqrt_neg, false)
- const_bool_nodef (has_nan, true)
- const_bool_nodef (has_infinity, true)
- const_bool_nodef (fpu_check_inexact, true)
- const_bool_nodef (fpu_check_nan_result, true)

Static Public Member Functions

- static void handle_result (Result r)

11.52.1 Detailed Description

Definition at line 34 of file Checked_Number.defs.hh.

11.52.2 Member Function Documentation

11.52.2.1 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (fpu_check_nan_result, true)

11.52.2.2 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (fpu_check_inexact, true)

11.52.2.3 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (has_infinity, true)

11.52.2.4 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (has_nan, true)

11.52.2.5 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_sqrt_neg, false)

11.52.2.6 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_inf_mod, false)

11.52.2.7 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_inf_div_inf, false)

11.52.2.8 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_div_zero, false)

11.52.2.9 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_inf_mul_zero, false)

11.52.2.10 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_inf_sub_inf, false)

11.52.2.11 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_inf_add_inf, false)

11.52.2.12 Parma_Polyhedra_Library::Extended_Number_Policy::const_bool_nodef (check_overflow, true)

11.52.2.13 void Parma_Polyhedra_Library::Extended_Number_Policy::handle_result (Result *r*) [inline, static]

Definition at line 65 of file Checked_Number.inlines.hh.

References Parma_Polyhedra_Library::throw_result_exception().

The documentation for this struct was generated from the following files:

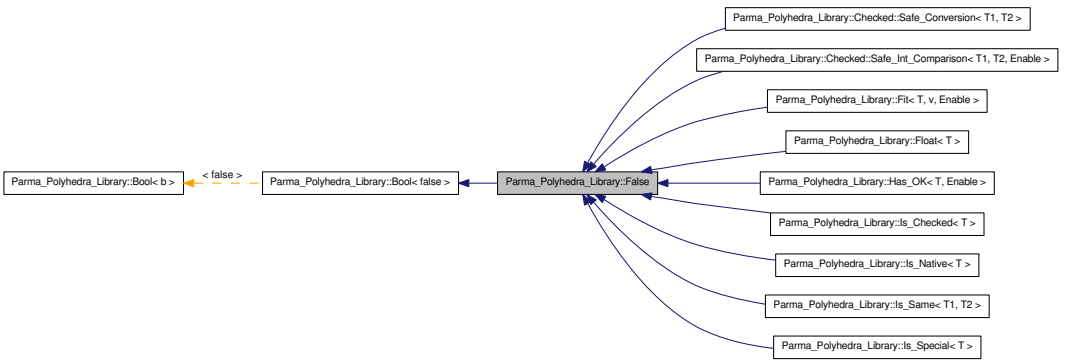
- Checked_Number.defs.hh
- Checked_Number.inlines.hh

11.53 Parma_Polyhedra_Library::False Struct Reference

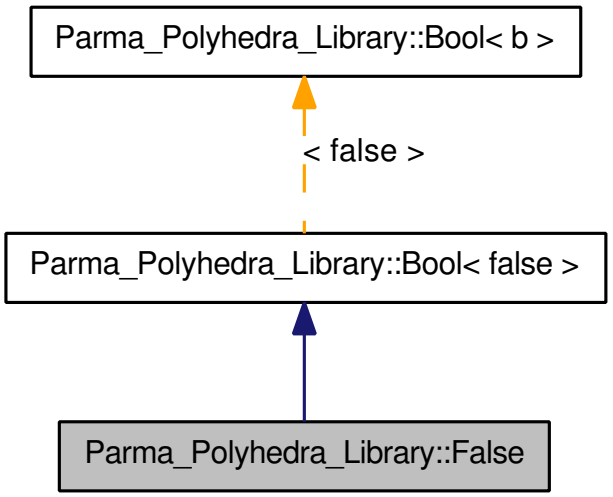
A class holding a constant called `value` that evaluates to `false`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::False:



Collaboration diagram for Parma_Polyhedra_Library::False:



11.53.1 Detailed Description

A class holding a constant called `value` that evaluates to `false`.

Definition at line 167 of file meta_programming.hh.

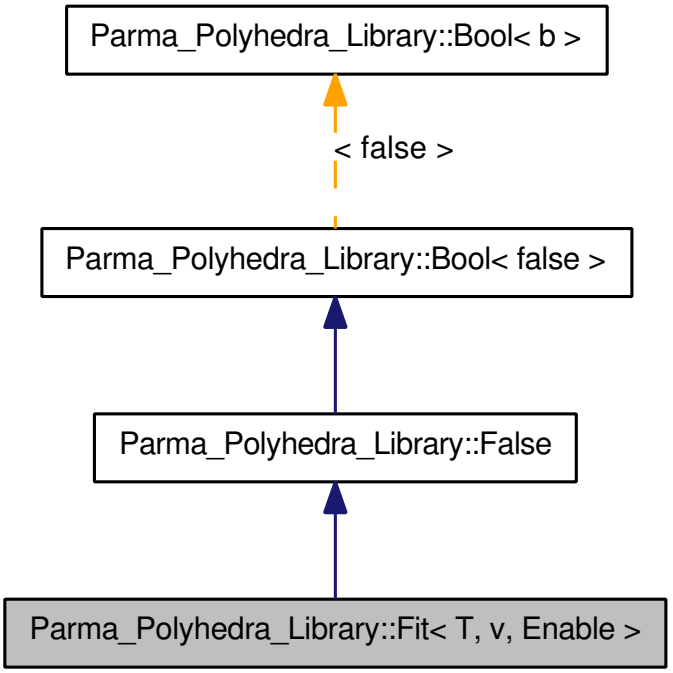
The documentation for this struct was generated from the following file:

- meta_programming.hh

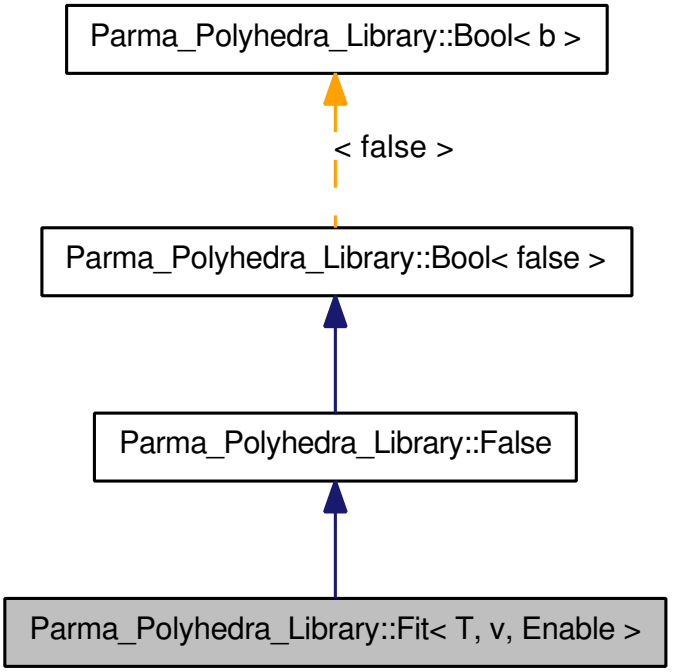
11.54 Parma_Polyhedra_Library::Fit< T, v, Enable > Struct Template Reference

```
#include <globals.defs.hh>
```

Inheritance diagram for Parma_Polyhedra_Library::Fit< T, v, Enable >:



Collaboration diagram for Parma_Polyhedra_Library::Fit< T, v, Enable >:



11.54.1 Detailed Description

template<typename T, long long v, typename Enable = void> struct Parma_Polyhedra_Library::Fit< T, v, Enable >

Definition at line 339 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

- globals.defs.hh

11.55 Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type > Struct Template Reference

`#include <globals.defs.hh>`

Public Types

- enum { value }

11.55.1 Detailed Description

template<typename T, long long v> struct Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type >

Definition at line 343 of file globals.defs.hh.

11.55.2 Member Enumeration Documentation

11.55.2.1 template<typename T , long long v> anonymous enum

Enumerator:

value

Definition at line 344 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

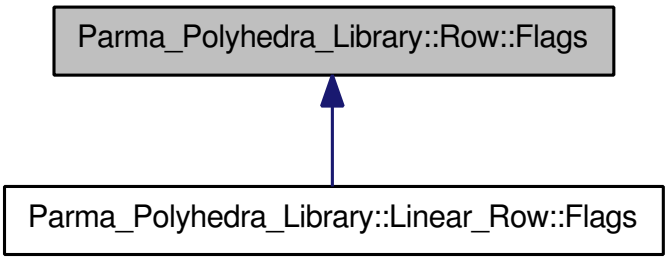
- globals.defs.hh

11.56 Parma_Polyhedra_Library::Row::Flags Class Reference

Wrapper class to represent a set of flags with bits in a native unsigned integral type.

`#include <Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Row::Flags:



Public Member Functions

- Flags ()

  Constructs an object with all the flags unset.

- bool operator== (const Flags &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are equal.*

- bool operator!= (const Flags &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Uses the ASCII Flags representation from `s` *to recreate* `*this`.

Protected Types

- typedef unsigned int base_type

  A native integral type holding the bits that encode the flags.

Protected Member Functions

- Flags (base_type n)

  Constructs an object with flags set as in `n`.

- base_type get_bits () const

  Returns the integer encoding `*this`.

- void set_bits (base_type mask)

  Sets the bits in `mask`.

- void reset_bits (base_type mask)

  Resets the bits in `mask`.

- bool test_bits (base_type mask) const

  Returns `true` *if and only if all the bits in* `mask` *are set.*

Static Protected Attributes

- static const unsigned first_free_bit = 0

  Index of the first bit derived classes can use.

- static const unsigned num_bits = std::numeric_limits<base_type>::digits

  Total number of bits that can be stored.

Private Attributes

- base_type bits

  The integer encoding `*this`.

Friends

- class Row

11.56.1 Detailed Description

Wrapper class to represent a set of flags with bits in a native unsigned integral type.

Definition at line 98 of file Row.defs.hh.

11.56.2 Member Typedef Documentation

11.56.2.1 typedef unsigned int Parma_Polyhedra_Library::Row::Flags::base_type [protected]

A native integral type holding the bits that encode the flags.

Definition at line 124 of file Row.defs.hh.

11.56.3 Constructor & Destructor Documentation

11.56.3.1 Parma_Polyhedra_Library::Row::Flags::Flags () [inline]

Constructs an object with all the flags unset.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 35 of file Row.inlines.hh.

11.56.3.2 Parma_Polyhedra_Library::Row::Flags::Flags (base_type *n*) [inline, explicit, protected]

Constructs an object with flags set as in `n`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 40 of file Row.inlines.hh.

11.56.4 Member Function Documentation

11.56.4.1 void Parma_Polyhedra_Library::Row::Flags::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 135 of file Row.cc.

References bits.

11.56.4.2 void Parma_Polyhedra_Library::Row::Flags::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Referenced by Parma_Polyhedra_Library::Row::ascii_dump().

11.56.4.3 bool Parma_Polyhedra_Library::Row::Flags::ascii_load (std::istream & *s*)

Uses the ASCII Flags representation from `s` to recreate *this.

Returns `true` if successful, `false` otherwise. The ASCII representation is as output by Parma_Polyhedra_Library::Row::Flags::ascii_dump().

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 149 of file Row.cc.

11.56.4.4 Row::Flags::base_type Parma_Polyhedra_Library::Row::Flags::get_bits () const [inline, protected]

Returns the integer encoding `*this`.

Definition at line 45 of file Row.inlines.hh.

References bits.

Referenced by operator==(), and Parma_Polyhedra_Library::Linear_Row::Flags::operator==().

11.56.4.5 bool Parma_Polyhedra_Library::Row::Flags::operator!= (const Flags & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 71 of file Row.inlines.hh.

References operator==().

11.56.4.6 bool Parma_Polyhedra_Library::Row::Flags::operator== (const Flags & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are equal.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 65 of file Row.inlines.hh.

References first_free_bit, and get_bits().

Referenced by operator!=().

11.56.4.7 void Parma_Polyhedra_Library::Row::Flags::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

11.56.4.8 void Parma_Polyhedra_Library::Row::Flags::reset_bits (base_type *mask*) [inline, protected]

Resets the bits in `mask`.

Definition at line 55 of file Row.inlines.hh.

References bits.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::set_is_line_or_equality(), and Parma_-Polyhedra_Library::Linear_Row::Flags::set_necessarily_closed().

11.56.4.9 void Parma_Polyhedra_Library::Row::Flags::set_bits (base_type *mask*) [inline, protected]

Sets the bits in `mask`.

Definition at line 50 of file Row.inlines.hh.

References bits.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::Flags(), Parma_Polyhedra_-Library::Linear_Row::Flags::set_is_line_or_equality(), Parma_Polyhedra_Library::Linear_-Row::Flags::set_is_ray_or_point_or_inequality(), Parma_Polyhedra_Library::Linear_Row::Flags::set_-necessarily_closed(), and Parma_Polyhedra_Library::Linear_Row::Flags::set_not_necessarily_closed().

11.56.4.10 bool Parma_Polyhedra_Library::Row::Flags::test_bits (base_type *mask*) const [inline, protected]

Returns `true` if and only if all the bits in `mask` are set.

Definition at line 60 of file Row.inlines.hh.

References bits.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::ascii_dump(), Parma_Polyhedra_-Library::Linear_Row::Flags::is_line_or_equality(), Parma_Polyhedra_Library::Linear_Row::Flags::is_-necessarily_closed(), Parma_Polyhedra_Library::Linear_Row::Flags::is_not_necessarily_closed(), and Parma_Polyhedra_Library::Linear_Row::Flags::is_ray_or_point_or_inequality().

11.56.5 Friends And Related Function Documentation

11.56.5.1 friend class Row [friend]

Definition at line 154 of file Row.defs.hh.

11.56.6 Member Data Documentation

11.56.6.1 base_type Parma_Polyhedra_Library::Row::Flags::bits [private]

The integer encoding `*this`.

Definition at line 152 of file Row.defs.hh.

Referenced by ascii_dump(), get_bits(), reset_bits(), set_bits(), and test_bits().

11.56.6.2 const unsigned Parma_Polyhedra_Library::Row::Flags::first_free_bit = 0 [static, protected]

Index of the first bit derived classes can use.

Reimplemented in Parma_Polyhedra_Library::Linear_Row::Flags.

Definition at line 127 of file Row.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_Row::Flags::ascii_load(), and operator==().

11.56.6.3 const unsigned Parma_Polyhedra_Library::Row::Flags::num_bits = std::numeric_limits<base_type>::digits [static, protected]

Total number of bits that can be stored.

Definition at line 130 of file Row.defs.hh.

The documentation for this class was generated from the following files:

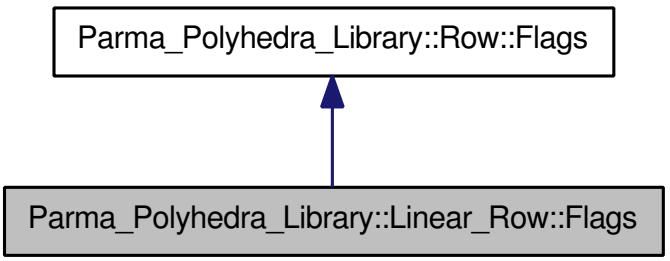
- Row.defs.hh
- Row.inlines.hh
- Row.cc

11.57 Parma_Polyhedra_Library::Linear_Row::Flags Class Reference

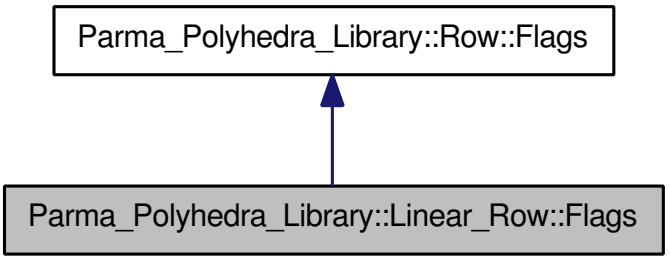
The type of the object to which the coefficients refer to, encoding both topology and kind.

`#include <Linear_Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Linear_Row::Flags:



Collaboration diagram for Parma_Polyhedra_Library::Linear_Row::Flags:



Public Member Functions

- Flags ()
  Default constructor: builds an object where all flags are invalid.
- Flags (Topology t)
  Builds an object corresponding to the topology `t`.
- Flags (Topology t, Kind k)
  Builds an object corresponding to the topology `t` *and kind* `k`.
- bool operator== (const Flags &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are equal.*
- bool operator!= (const Flags &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are different.*
- void ascii_dump () const
  Writes to `std::cerr` *an ASCII representation of* `*this`.
- void ascii_dump (std::ostream &s) const
  Writes to `s` *an ASCII representation of* `*this`.
- void print () const
  Prints `*this` *to* `std::cerr` *using* `operator<<`.
- bool ascii_load (std::istream &s)
  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

Testing and setting the type

- Topology topology () const
- bool is_necessarily_closed () const
- bool is_not_necessarily_closed () const
- bool is_line_or_equality () const
- bool is_ray_or_point_or_inequality () const
- void set_necessarily_closed ()
- void set_not_necessarily_closed ()
- void set_is_line_or_equality ()
- void set_is_ray_or_point_or_inequality ()

Static Protected Attributes

- static const unsigned first_free_bit = Row::Flags::first_free_bit + 4
  Index of the first bit derived classes can use.

Private Member Functions

- Flags (base_type mask)
  Builds the type from a bit-mask.

Static Private Attributes

The bits that are currently in use

- static const unsigned rpi_validity_bit = Row::Flags::first_free_bit + 0
- static const unsigned rpi_bit = Row::Flags::first_free_bit + 1
- static const unsigned nnc_validity_bit = Row::Flags::first_free_bit + 2
- static const unsigned nnc_bit = Row::Flags::first_free_bit + 3

Friends

- class Parma_Polyhedra_Library::Linear_Row

11.57.1 Detailed Description

The type of the object to which the coefficients refer to, encoding both topology and kind. This combines the information about the topology (necessarily closed or not) and the kind (line/equality or ray/point/inequality) of a Linear_Row object.

Definition at line 144 of file Linear_Row.defs.hh.

11.57.2 Constructor & Destructor Documentation

11.57.2.1 Parma_Polyhedra_Library::Linear_Row::Flags::Flags () [inline]

Default constructor: builds an object where all flags are invalid.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 34 of file Linear_Row.inlines.hh.

11.57.2.2 Parma_Polyhedra_Library::Linear_Row::Flags::Flags (Topology *t*) [inline, explicit]

Builds an object corresponding to the topology `t`.

Definition at line 40 of file Linear_Row.inlines.hh.

References nnc_validity_bit, and Parma_Polyhedra_Library::Row::Flags::set_bits().

11.57.2.3 Parma_Polyhedra_Library::Linear_Row::Flags::Flags (Topology *t*, Kind *k*) [inline]

Builds an object corresponding to the topology `t` and kind `k`.

Definition at line 48 of file Linear_Row.inlines.hh.

References nnc_validity_bit, rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::set_bits().

11.57.2.4 Parma_Polyhedra_Library::Linear_Row::Flags::Flags (base_type *mask*) [explicit, private]

Builds the type from a bit-mask.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

11.57.3 Member Function Documentation

11.57.3.1 void Parma_Polyhedra_Library::Linear_Row::Flags::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 155 of file Linear_Row.cc.

References nnc_bit, nnc_validity_bit, rpi_bit, rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::test_bits().

11.57.3.2 void Parma_Polyhedra_Library::Linear_Row::Flags::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Referenced by Parma_Polyhedra_Library::Linear_Row::ascii_dump().

11.57.3.3 bool Parma_Polyhedra_Library::Linear_Row::Flags::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 170 of file Linear_Row.cc.

References Parma_Polyhedra_Library::Row::Flags::first_free_bit, Parma_Polyhedra_Library::Interval_NS::reset_bits(), and Parma_Polyhedra_Library::Interval_NS::set_bits().

11.57.3.4 bool Parma_Polyhedra_Library::Linear_Row::Flags::is_line_or_equality () const [inline]

Definition at line 71 of file Linear_Row.inlines.hh.

References is_ray_or_point_or_inequality(), rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::test_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::is_line_or_equality().

11.57.3.5 bool Parma_Polyhedra_Library::Linear_Row::Flags::is_necessarily_closed () const [inline]

Definition at line 91 of file Linear_Row.inlines.hh.

References is_not_necessarily_closed(), nnc_validity_bit, and Parma_Polyhedra_Library::Row::Flags::test_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), and topology().

11.57.3.6 bool Parma_Polyhedra_Library::Linear_Row::Flags::is_not_necessarily_closed () const [inline]

Definition at line 85 of file Linear_Row.inlines.hh.

References nnc_bit, nnc_validity_bit, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, and Parma_Polyhedra_Library::Row::Flags::test_bits().

Referenced by is_necessarily_closed().

11.57.3.7 bool Parma_Polyhedra_Library::Linear_Row::Flags::is_ray_or_point_or_inequality () const [inline]

Definition at line 57 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, rpi_bit, rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::test_bits().

Referenced by is_line_or_equality(), and Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality().

11.57.3.8 bool Parma_Polyhedra_Library::Linear_Row::Flags::operator!= (const Flags & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 124 of file Linear_Row.inlines.hh.

References operator==().

11.57.3.9 bool Parma_Polyhedra_Library::Linear_Row::Flags::operator== (const Flags & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are equal.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 118 of file Linear_Row.inlines.hh.

References first_free_bit, and Parma_Polyhedra_Library::Row::Flags::get_bits().

Referenced by operator!=().

11.57.3.10 void Parma_Polyhedra_Library::Linear_Row::Flags::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

11.57.3.11 void Parma_Polyhedra_Library::Linear_Row::Flags::set_is_line_or_equality () [inline]

Definition at line 77 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, Parma_Polyhedra_Library::Row::Flags::reset_bits(), rpi_bit, rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::set_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality().

11.57.3.12 void Parma_Polyhedra_Library::Linear_Row::Flags::set_is_ray_or_point_or_inequality () [inline]

Definition at line 63 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, rpi_bit, rpi_validity_bit, and Parma_Polyhedra_Library::Row::Flags::set_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::set_is_ray_or_point_or_inequality().

11.57.3.13 void Parma_Polyhedra_Library::Linear_Row::Flags::set_necessarily_closed () [inline]

Definition at line 105 of file Linear_Row.inlines.hh.

References nnc_bit, nnc_validity_bit, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Row::Flags::reset_bits(), and Parma_Polyhedra_Library::Row::Flags::set_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::set_necessarily_closed().

11.57.3.14 void Parma_Polyhedra_Library::Linear_Row::Flags::set_not_necessarily_closed () [inline]

Definition at line 97 of file Linear_Row.inlines.hh.

References nnc_bit, nnc_validity_bit, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, and Parma_Polyhedra_Library::Row::Flags::set_bits().

Referenced by Parma_Polyhedra_Library::Linear_Row::set_not_necessarily_closed().

11.57.3.15 Topology Parma_Polyhedra_Library::Linear_Row::Flags::topology () const [inline]

Definition at line 113 of file Linear_Row.inlines.hh.

References is_necessarily_closed(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, and Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED.

Referenced by Parma_Polyhedra_Library::Linear_Row::topology().

11.57.4 Friends And Related Function Documentation

11.57.4.1 friend class Parma_Polyhedra_Library::Linear_Row [friend]

Definition at line 209 of file Linear_Row.defs.hh.

11.57.5 Member Data Documentation

11.57.5.1 const unsigned Parma_Polyhedra_Library::Linear_Row::Flags::first_free_bit = Row::Flags::first_free_bit + 4 [static, protected]

Index of the first bit derived classes can use.

Reimplemented from Parma_Polyhedra_Library::Row::Flags.

Definition at line 207 of file Linear_Row.defs.hh.

Referenced by operator==().

11.57.5.2 const unsigned Parma_Polyhedra_Library::Linear_Row::Flags::nnc_bit = Row::Flags::first_free_bit + 3 [static, private]

Definition at line 201 of file Linear_Row.defs.hh.

Referenced by ascii_dump(), is_not_necessarily_closed(), set_necessarily_closed(), and set_not_necessarily_closed().

11.57.5.3 const unsigned Parma_Polyhedra_Library::Linear_Row::Flags::nnc_validity_bit = Row::Flags::first_free_bit + 2 [static, private]

Definition at line 199 of file Linear_Row.defs.hh.

Referenced by ascii_dump(), Flags(), is_necessarily_closed(), is_not_necessarily_closed(), set_necessarily_closed(), and set_not_necessarily_closed().

11.57.5.4 const unsigned Parma_Polyhedra_Library::Linear_Row::Flags::rpi_bit = Row::Flags::first_free_bit + 1 [static, private]

Definition at line 197 of file Linear_Row.defs.hh.

Referenced by ascii_dump(), is_ray_or_point_or_inequality(), set_is_line_or_equality(), and set_is_ray_or_point_or_inequality().

11.57.5.5 const unsigned Parma_Polyhedra_Library::Linear_Row::Flags::rpi_validity_bit = Row::Flags::first_free_bit + 0 [static, private]

Definition at line 195 of file Linear_Row.defs.hh.

Referenced by ascii_dump(), Flags(), is_line_or_equality(), is_ray_or_point_or_inequality(), set_is_line_or_equality(), and set_is_ray_or_point_or_inequality().

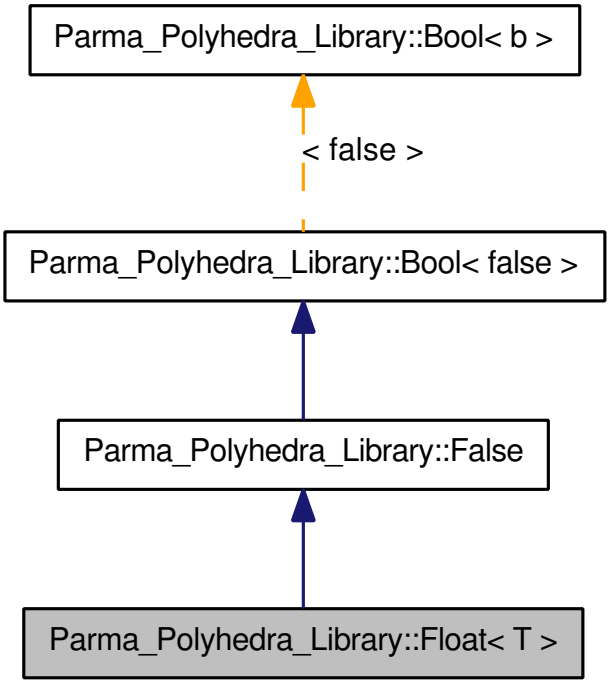
The documentation for this class was generated from the following files:

- Linear_Row.defs.hh
- Linear_Row.inlines.hh
- Linear_Row.cc

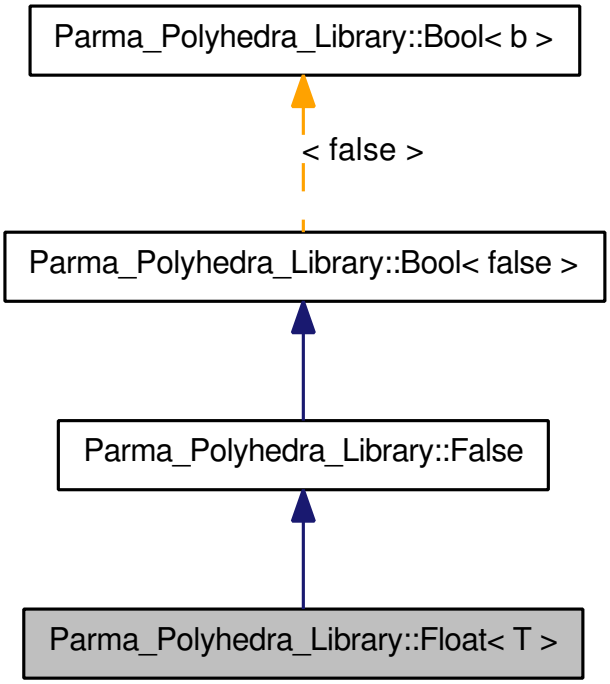
11.58 Parma_Polyhedra_Library::Float< T > Class Template Reference

`#include <Float.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Float< T >:



Collaboration diagram for Parma_Polyhedra_Library::Float< T >:



11.58.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::Float< T >

Definition at line 274 of file Float.defs.hh.

The documentation for this class was generated from the following file:

- Float.defs.hh

11.59 Parma_Polyhedra_Library::Checked::Float_2exp Struct Reference

`#include <checked_float.inlines.hh>`

Public Member Functions

- const_bool_nodef (has_nan, false)
- const_bool_nodef (has_infinity, false)

11.59.1 Detailed Description

Definition at line 595 of file checked_float.inlines.hh.

11.59.2 Member Function Documentation

11.59.2.1 Parma_Polyhedra_Library::Checked::Float_2exp::const_bool_nodef (has_infinity, false)

11.59.2.2 Parma_Polyhedra_Library::Checked::Float_2exp::const_bool_nodef (has_nan, false)

The documentation for this struct was generated from the following file:

- checked_float.inlines.hh

11.60 Parma_Polyhedra_Library::float_ibm_double Struct Reference

`#include <Float.defs.hh>`

Static Public Attributes

- static const unsigned int BASE = 16
- static const unsigned int EXPONENT_BITS = 7
- static const unsigned int MANTISSA_BITS = 56
- static const int EXPONENT_BIAS = 64

11.60.1 Detailed Description

Definition at line 184 of file Float.defs.hh.

11.60.2 Member Data Documentation

11.60.2.1 const unsigned int Parma_Polyhedra_Library::float_ibm_double::BASE = 16 [static]

Definition at line 185 of file Float.defs.hh.

11.60.2.2 const int Parma_Polyhedra_Library::float_ibm_double::EXPONENT_BIAS = 64 [static]

Definition at line 188 of file Float.defs.hh.

11.60.2.3 const unsigned int Parma_Polyhedra_Library::float_ibm_double::EXPONENT_BITS = 7 [static]

Definition at line 186 of file Float.defs.hh.

11.60.2.4 const unsigned int Parma_Polyhedra_Library::float_ibm_double::MANTISSA_BITS = 56 [static]

Definition at line 187 of file Float.defs.hh.

The documentation for this struct was generated from the following file:

- Float.defs.hh

11.61 Parma_Polyhedra_Library::float_ibm_single Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()
- void inc ()

- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint32_t word

Static Public Attributes

- static const uint32_t SGN_MASK = 0x80000000
- static const uint32_t EXP_MASK = 0x7f000000
- static const uint32_t POS_INF = 0x7f000000
- static const uint32_t NEG_INF = 0xff000000
- static const uint32_t POS_ZERO = 0x00000000
- static const uint32_t NEG_ZERO = 0x80000000
- static const unsigned int BASE = 16
- static const unsigned int EXPONENT_BITS = 7
- static const unsigned int MANTISSA_BITS = 24
- static const int EXPONENT_BIAS = 64
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.61.1 Detailed Description

Definition at line 153 of file Float.defs.hh.

11.61.2 Member Function Documentation

11.61.2.1 void Parma_Polyhedra_Library::float_ibm_single::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 282 of file Float.inlines.hh.

References EXPONENT_BIAS, MANTISSA_BITS, SGN_MASK, and word.

11.61.2.2 void Parma_Polyhedra_Library::float_ibm_single::dec () [inline]

Definition at line 265 of file Float.inlines.hh.

References word.

11.61.2.3 void Parma_Polyhedra_Library::float_ibm_single::inc () [inline]

Definition at line 270 of file Float.inlines.hh.

References word.

11.61.2.4 int Parma_Polyhedra_Library::float_ibm_single::is_inf () const [inline]

Definition at line 232 of file Float.inlines.hh.

References NEG_INF, POS_INF, and word.

11.61.2.5 int Parma_Polyhedra_Library::float_ibm_single::is_nan () const [inline]

Definition at line 241 of file Float.inlines.hh.

References POS_INF, SGN_MASK, and word.

11.61.2.6 int Parma_Polyhedra_Library::float_ibm_single::is_zero () const [inline]

Definition at line 246 of file Float.inlines.hh.

References NEG_ZERO, POS_ZERO, and word.

11.61.2.7 void Parma_Polyhedra_Library::float_ibm_single::negate () [inline]

Definition at line 255 of file Float.inlines.hh.

References SGN_MASK, and word.

11.61.2.8 void Parma_Polyhedra_Library::float_ibm_single::set_max (bool *negative*) [inline]

Definition at line 275 of file Float.inlines.hh.

References SGN_MASK, and word.

11.61.2.9 int Parma_Polyhedra_Library::float_ibm_single::sign_bit () const [inline]

Definition at line 260 of file Float.inlines.hh.

References SGN_MASK, and word.

11.61.3 Member Data Documentation

11.61.3.1 const unsigned int Parma_Polyhedra_Library::float_ibm_single::BASE = 16 [static]

Definition at line 161 of file Float.defs.hh.

11.61.3.2 const uint32_t Parma_Polyhedra_Library::float_ibm_single::EXP_MASK = 0x7f000000 [static]

Definition at line 156 of file Float.defs.hh.

11.61.3.3 const int Parma_Polyhedra_Library::float_ibm_single::EXPONENT_BIAS = 64 [static]

Definition at line 164 of file Float.defs.hh.

Referenced by build().

11.61.3.4 const unsigned int Parma_Polyhedra_Library::float_ibm_single::EXPONENT_BITS = 7 [static]

Definition at line 162 of file Float.defs.hh.

11.61.3.5 const int Parma_Polyhedra_Library::float_ibm_single::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 165 of file Float.defs.hh.

11.61.3.6 const int Parma_Polyhedra_Library::float_ibm_single::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 166 of file Float.defs.hh.

11.61.3.7 const int Parma_Polyhedra_Library::float_ibm_single::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 167 of file Float.defs.hh.

11.61.3.8 const unsigned int Parma_Polyhedra_Library::float_ibm_single::MANTISSA_BITS = 24 [static]

Definition at line 163 of file Float.defs.hh.

Referenced by build().

11.61.3.9 const uint32_t Parma_Polyhedra_Library::float_ibm_single::NEG_INF = 0xff000000 [static]

Definition at line 158 of file Float.defs.hh.

Referenced by is_inf().

11.61.3.10 const uint32_t Parma_Polyhedra_Library::float_ibm_single::NEG_ZERO = 0x80000000 [static]

Definition at line 160 of file Float.defs.hh.

Referenced by is_zero().

11.61.3.11 const uint32_t Parma_Polyhedra_Library::float_ibm_single::POS_INF = 0x7f000000 [static]

Definition at line 157 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.61.3.12 const uint32_t Parma_Polyhedra_Library::float_ibm_single::POS_ZERO = 0x00000000 [static]

Definition at line 159 of file Float.defs.hh.

Referenced by is_zero().

11.61.3.13 const uint32_t Parma_Polyhedra_Library::float_ibm_single::SGN_MASK = 0x80000000 [static]

Definition at line 155 of file Float.defs.hh.

Referenced by build(), is_nan(), negate(), set_max(), and sign_bit().

11.61.3.14 uint32_t Parma_Polyhedra_Library::float_ibm_single::word

Definition at line 154 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh

11.62 Parma_Polyhedra_Library::float_ieee754_double Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()
- void inc ()
- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint32_t lsp
- uint32_t msp

Static Public Attributes

- static const uint32_t MSP_SGN_MASK = 0x80000000
- static const uint32_t MSP_POS_INF = 0x7ff00000
- static const uint32_t MSP_NEG_INF = 0xfff00000
- static const uint32_t MSP_POS_ZERO = 0x00000000
- static const uint32_t MSP_NEG_ZERO = 0x80000000
- static const uint32_t LSP_INF = 0
- static const uint32_t LSP_ZERO = 0
- static const uint32_t LSP_MAX = 0xffffffff
- static const unsigned int BASE = 2
- static const unsigned int EXPONENT_BITS = 11
- static const unsigned int MANTISSA_BITS = 52
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_BIAS = EXPONENT_MAX
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.62.1 Detailed Description

Definition at line 114 of file Float.defs.hh.

11.62.2 Member Function Documentation

11.62.2.1 void Parma_Polyhedra_Library::float_ieee754_double::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 214 of file Float.inlines.hh.

References EXPONENT_BIAS, lsp, MANTISSA_BITS, msp, and MSP_SGN_MASK.

11.62.2.2 void Parma_Polyhedra_Library::float_ieee754_double::dec () [inline]

Definition at line 186 of file Float.inlines.hh.

References lsp, LSP_MAX, and msp.

11.62.2.3 void Parma_Polyhedra_Library::float_ieee754_double::inc () [inline]

Definition at line 196 of file Float.inlines.hh.

References lsp, LSP_MAX, and msp.

11.62.2.4 int Parma_Polyhedra_Library::float_ieee754_double::is_inf () const [inline]

Definition at line 148 of file Float.inlines.hh.

References lsp, LSP_INF, msp, MSP_NEG_INF, and MSP_POS_INF.

11.62.2.5 int Parma_Polyhedra_Library::float_ieee754_double::is_nan () const [inline]

Definition at line 159 of file Float.inlines.hh.

References lsp, LSP_INF, msp, MSP_POS_INF, and MSP_SGN_MASK.

11.62.2.6 int Parma_Polyhedra_Library::float_ieee754_double::is_zero () const [inline]

Definition at line 165 of file Float.inlines.hh.

References lsp, LSP_ZERO, msp, MSP_NEG_ZERO, and MSP_POS_ZERO.

11.62.2.7 void Parma_Polyhedra_Library::float_ieee754_double::negate () [inline]

Definition at line 176 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.62.2.8 void Parma_Polyhedra_Library::float_ieee754_double::set_max (bool *negative*) [inline]

Definition at line 206 of file Float.inlines.hh.

References lsp, msp, and MSP_SGN_MASK.

11.62.2.9 int Parma_Polyhedra_Library::float_ieee754_double::sign_bit () const [inline]

Definition at line 181 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.62.3 Member Data Documentation

11.62.3.1 const unsigned int Parma_Polyhedra_Library::float_ieee754_double::BASE = 2 [static]

Definition at line 130 of file Float.defs.hh.

11.62.3.2 const int Parma_Polyhedra_Library::float_ieee754_double::EXPONENT_BIAS = EXPONENT_MAX [static]

Definition at line 134 of file Float.defs.hh.

Referenced by build().

11.62.3.3 const unsigned int Parma_Polyhedra_Library::float_ieee754_double::EXPONENT_BITS = 11 [static]

Definition at line 131 of file Float.defs.hh.

11.62.3.4 const int Parma_Polyhedra_Library::float_ieee754_double::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 133 of file Float.defs.hh.

11.62.3.5 const int Parma_Polyhedra_Library::float_ieee754_double::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 135 of file Float.defs.hh.

11.62.3.6 const int Parma_Polyhedra_Library::float_ieee754_double::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 136 of file Float.defs.hh.

11.62.3.7 uint32_t Parma_Polyhedra_Library::float_ieee754_double::lsp

Definition at line 119 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), and set_max().

11.62.3.8 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::LSP_INF = 0 [static]

Definition at line 127 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.62.3.9 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::LSP_MAX = 0xffffffff [static]

Definition at line 129 of file Float.defs.hh.

Referenced by dec(), and inc().

11.62.3.10 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::LSP_ZERO = 0 [static]

Definition at line 128 of file Float.defs.hh.

Referenced by is_zero().

11.62.3.11 const unsigned int Parma_Polyhedra_Library::float_ieee754_double::MANTISSA_BITS = 52 [static]

Definition at line 132 of file Float.defs.hh.

Referenced by build().

11.62.3.12 uint32_t Parma_Polyhedra_Library::float_ieee754_double::msp

Definition at line 120 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

11.62.3.13 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::MSP_NEG_INF = 0xfff00000 [static]

Definition at line 124 of file Float.defs.hh.

Referenced by is_inf().

11.62.3.14 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::MSP_NEG_ZERO = 0x80000000 [static]

Definition at line 126 of file Float.defs.hh.

Referenced by is_zero().

11.62.3.15 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::MSP_POS_INF = 0x7ff00000 [static]

Definition at line 123 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.62.3.16 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::MSP_POS_ZERO = 0x00000000 [static]

Definition at line 125 of file Float.defs.hh.

Referenced by is_zero().

11.62.3.17 const uint32_t Parma_Polyhedra_Library::float_ieee754_double::MSP_SGN_MASK = 0x80000000 [static]

Definition at line 122 of file Float.defs.hh.

Referenced by build(), is_nan(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh

11.63 Parma_Polyhedra_Library::float_ieee754_half Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()
- void inc ()
- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint16_t word

Static Public Attributes

- static const uint16_t SGN_MASK = 0x8000
- static const uint16_t EXP_MASK = 0xfc00
- static const uint16_t POS_INF = 0xfc00
- static const uint16_t NEG_INF = 0x7c00
- static const uint16_t POS_ZERO = 0x0000
- static const uint16_t NEG_ZERO = 0x8000
- static const unsigned int BASE = 2
- static const unsigned int EXPONENT_BITS = 5
- static const unsigned int MANTISSA_BITS = 10
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_BIAS = EXPONENT_MAX
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.63.1 Detailed Description

Definition at line 45 of file Float.defs.hh.

11.63.2 Member Function Documentation

11.63.2.1 void Parma_Polyhedra_Library::float_ieee754_half::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 82 of file Float.inlines.hh.

References EXPONENT_BIAS, MANTISSA_BITS, SGN_MASK, and word.

11.63.2.2 void Parma_Polyhedra_Library::float_ieee754_half::dec () [inline]

Definition at line 65 of file Float.inlines.hh.

References word.

11.63.2.3 void Parma_Polyhedra_Library::float_ieee754_half::inc () [inline]

Definition at line 70 of file Float.inlines.hh.

References word.

11.63.2.4 int Parma_Polyhedra_Library::float_ieee754_half::is_inf () const [inline]

Definition at line 32 of file Float.inlines.hh.

References NEG_INF, POS_INF, and word.

11.63.2.5 int Parma_Polyhedra_Library::float_ieee754_half::is_nan () const [inline]

Definition at line 41 of file Float.inlines.hh.

References POS_INF, SGN_MASK, and word.

11.63.2.6 int Parma_Polyhedra_Library::float_ieee754_half::is_zero () const [inline]

Definition at line 46 of file Float.inlines.hh.

References NEG_ZERO, POS_ZERO, and word.

11.63.2.7 void Parma_Polyhedra_Library::float_ieee754_half::negate () [inline]

Definition at line 55 of file Float.inlines.hh.

References SGN_MASK, and word.

11.63.2.8 void Parma_Polyhedra_Library::float_ieee754_half::set_max (bool *negative*) [inline]

Definition at line 75 of file Float.inlines.hh.

References SGN_MASK, and word.

11.63.2.9 int Parma_Polyhedra_Library::float_ieee754_half::sign_bit () const [inline]

Definition at line 60 of file Float.inlines.hh.

References SGN_MASK, and word.

11.63.3 Member Data Documentation

11.63.3.1 const unsigned int Parma_Polyhedra_Library::float_ieee754_half::BASE = 2 [static]

Definition at line 53 of file Float.defs.hh.

11.63.3.2 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::EXP_MASK = 0xfc00 [static]

Definition at line 48 of file Float.defs.hh.

11.63.3.3 const int Parma_Polyhedra_Library::float_ieee754_half::EXPONENT_BIAS = EXPONENT_MAX [static]

Definition at line 57 of file Float.defs.hh.

Referenced by build().

11.63.3.4 const unsigned int Parma_Polyhedra_Library::float_ieee754_half::EXPONENT_BITS = 5 [static]

Definition at line 54 of file Float.defs.hh.

11.63.3.5 const int Parma_Polyhedra_Library::float_ieee754_half::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 56 of file Float.defs.hh.

11.63.3.6 const int Parma_Polyhedra_Library::float_ieee754_half::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 58 of file Float.defs.hh.

11.63.3.7 const int Parma_Polyhedra_Library::float_ieee754_half::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 59 of file Float.defs.hh.

11.63.3.8 const unsigned int Parma_Polyhedra_Library::float_ieee754_half::MANTISSA_BITS = 10 [static]

Definition at line 55 of file Float.defs.hh.

Referenced by build().

11.63.3.9 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::NEG_INF = 0x7c00 [static]

Definition at line 50 of file Float.defs.hh.

Referenced by is_inf().

11.63.3.10 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::NEG_ZERO = 0x8000 [static]

Definition at line 52 of file Float.defs.hh.

Referenced by is_zero().

11.63.3.11 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::POS_INF = 0xfc00 [static]

Definition at line 49 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.63.3.12 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::POS_ZERO = 0x0000 [static]

Definition at line 51 of file Float.defs.hh.

Referenced by is_zero().

11.63.3.13 const uint16_t Parma_Polyhedra_Library::float_ieee754_half::SGN_MASK = 0x8000 [static]

Definition at line 47 of file Float.defs.hh.

Referenced by build(), is_nan(), negate(), set_max(), and sign_bit().

11.63.3.14 uint16_t Parma_Polyhedra_Library::float_ieee754_half::word

Definition at line 46 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh

11.64 Parma_Polyhedra_Library::float_ieee754_quad Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()
- void inc ()
- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint64_t lsp
- uint64_t msp

Static Public Attributes

- static const uint64_t MSP_SGN_MASK = 0x8000000000000000ULL
- static const uint64_t MSP_POS_INF = 0x7fff000000000000ULL
- static const uint64_t MSP_NEG_INF = 0xffff000000000000ULL
- static const uint64_t MSP_POS_ZERO = 0x0000000000000000ULL
- static const uint64_t MSP_NEG_ZERO = 0x8000000000000000ULL
- static const uint64_t LSP_INF = 0
- static const uint64_t LSP_ZERO = 0
- static const uint64_t LSP_MAX = 0xffffffffffffffffULL
- static const unsigned int BASE = 2
- static const unsigned int EXPONENT_BITS = 15
- static const unsigned int MANTISSA_BITS = 112
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_BIAS = EXPONENT_MAX
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.64.1 Detailed Description

Definition at line 235 of file Float.defs.hh.

11.64.2 Member Function Documentation

11.64.2.1 void Parma_Polyhedra_Library::float_ieee754_quad::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 436 of file Float.inlines.hh.

References EXPONENT_BIAS, lsp, MANTISSA_BITS, msp, and MSP_SGN_MASK.

11.64.2.2 void Parma_Polyhedra_Library::float_ieee754_quad::dec () [inline]

Definition at line 408 of file Float.inlines.hh.

References lsp, LSP_MAX, and msp.

11.64.2.3 void Parma_Polyhedra_Library::float_ieee754_quad::inc () [inline]

Definition at line 418 of file Float.inlines.hh.

References lsp, LSP_MAX, and msp.

11.64.2.4 int Parma_Polyhedra_Library::float_ieee754_quad::is_inf () const [inline]

Definition at line 370 of file Float.inlines.hh.

References lsp, LSP_INF, msp, MSP_NEG_INF, and MSP_POS_INF.

11.64.2.5 int Parma_Polyhedra_Library::float_ieee754_quad::is_nan () const [inline]

Definition at line 381 of file Float.inlines.hh.

References lsp, LSP_INF, msp, MSP_POS_INF, and MSP_SGN_MASK.

11.64.2.6 int Parma_Polyhedra_Library::float_ieee754_quad::is_zero () const [inline]

Definition at line 387 of file Float.inlines.hh.

References lsp, LSP_ZERO, msp, MSP_NEG_ZERO, and MSP_POS_ZERO.

11.64.2.7 void Parma_Polyhedra_Library::float_ieee754_quad::negate () [inline]

Definition at line 398 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.64.2.8 void Parma_Polyhedra_Library::float_ieee754_quad::set_max (bool *negative*) [inline]

Definition at line 428 of file Float.inlines.hh.

References lsp, msp, and MSP_SGN_MASK.

11.64.2.9 int Parma_Polyhedra_Library::float_ieee754_quad::sign_bit () const [inline]

Definition at line 403 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.64.3 Member Data Documentation

11.64.3.1 const unsigned int Parma_Polyhedra_Library::float_ieee754_quad::BASE = 2 [static]

Definition at line 251 of file Float.defs.hh.

11.64.3.2 const int Parma_Polyhedra_Library::float_ieee754_quad::EXPONENT_BIAS = EXPONENT_MAX [static]

Definition at line 255 of file Float.defs.hh.

Referenced by build().

11.64.3.3 const unsigned int Parma_Polyhedra_Library::float_ieee754_quad::EXPONENT_BITS = 15 [static]

Definition at line 252 of file Float.defs.hh.

11.64.3.4 const int Parma_Polyhedra_Library::float_ieee754_quad::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 254 of file Float.defs.hh.

11.64.3.5 const int Parma_Polyhedra_Library::float_ieee754_quad::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 256 of file Float.defs.hh.

11.64.3.6 const int Parma_Polyhedra_Library::float_ieee754_quad::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 257 of file Float.defs.hh.

11.64.3.7 uint64_t Parma_Polyhedra_Library::float_ieee754_quad::lsp

Definition at line 240 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), and set_max().

11.64.3.8 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::LSP_INF = 0 [static]

Definition at line 248 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.64.3.9 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::LSP_MAX = 0xffffffffffffffffULL [static]

Definition at line 250 of file Float.defs.hh.

Referenced by dec(), and inc().

11.64.3.10 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::LSP_ZERO = 0 [static]

Definition at line 249 of file Float.defs.hh.

Referenced by is_zero().

11.64.3.11 const unsigned int Parma_Polyhedra_Library::float_ieee754_quad::MANTISSA_BITS = 112 [static]

Definition at line 253 of file Float.defs.hh.

Referenced by build().

11.64.3.12 uint64_t Parma_Polyhedra_Library::float_ieee754_quad::msp

Definition at line 241 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

11.64.3.13 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::MSP_NEG_INF = 0xffff000000000000ULL [static]

Definition at line 245 of file Float.defs.hh.

Referenced by is_inf().

11.64.3.14 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::MSP_NEG_ZERO = 0x8000000000000000ULL [static]

Definition at line 247 of file Float.defs.hh.

Referenced by is_zero().

11.64.3.15 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::MSP_POS_INF = 0x7fff000000000000ULL [static]

Definition at line 244 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.64.3.16 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::MSP_POS_ZERO = 0x0000000000000000ULL [static]

Definition at line 246 of file Float.defs.hh.

Referenced by is_zero().

11.64.3.17 const uint64_t Parma_Polyhedra_Library::float_ieee754_quad::MSP_SGN_MASK = 0x8000000000000000ULL [static]

Definition at line 243 of file Float.defs.hh.

Referenced by build(), is_nan(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh
- Float.cc

11.65 Parma_Polyhedra_Library::float_ieee754_single Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()

- void inc ()
- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint32_t word

Static Public Attributes

- static const uint32_t SGN_MASK = 0x80000000
- static const uint32_t EXP_MASK = 0x7f800000
- static const uint32_t POS_INF = 0x7f800000
- static const uint32_t NEG_INF = 0xff800000
- static const uint32_t POS_ZERO = 0x00000000
- static const uint32_t NEG_ZERO = 0x80000000
- static const unsigned int BASE = 2
- static const unsigned int EXPONENT_BITS = 8
- static const unsigned int MANTISSA_BITS = 23
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_BIAS = EXPONENT_MAX
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.65.1 Detailed Description

Definition at line 77 of file Float.defs.hh.

11.65.2 Member Function Documentation

11.65.2.1 void Parma_Polyhedra_Library::float_ieee754_single::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 140 of file Float.inlines.hh.

References EXPONENT_BIAS, MANTISSA_BITS, SGN_MASK, and word.

11.65.2.2 void Parma_Polyhedra_Library::float_ieee754_single::dec () [inline]

Definition at line 123 of file Float.inlines.hh.

References word.

11.65.2.3 void Parma_Polyhedra_Library::float_ieee754_single::inc () [inline]

Definition at line 128 of file Float.inlines.hh.

References word.

11.65.2.4 int Parma_Polyhedra_Library::float_ieee754_single::is_inf () const [inline]

Definition at line 90 of file Float.inlines.hh.

References NEG_INF, POS_INF, and word.

11.65.2.5 int Parma_Polyhedra_Library::float_ieee754_single::is_nan () const [inline]

Definition at line 99 of file Float.inlines.hh.

References POS_INF, SGN_MASK, and word.

11.65.2.6 int Parma_Polyhedra_Library::float_ieee754_single::is_zero () const [inline]

Definition at line 104 of file Float.inlines.hh.

References NEG_ZERO, POS_ZERO, and word.

11.65.2.7 void Parma_Polyhedra_Library::float_ieee754_single::negate () [inline]

Definition at line 113 of file Float.inlines.hh.

References SGN_MASK, and word.

11.65.2.8 void Parma_Polyhedra_Library::float_ieee754_single::set_max (bool *negative*) [inline]

Definition at line 133 of file Float.inlines.hh.

References SGN_MASK, and word.

11.65.2.9 int Parma_Polyhedra_Library::float_ieee754_single::sign_bit () const [inline]

Definition at line 118 of file Float.inlines.hh.

References SGN_MASK, and word.

11.65.3 Member Data Documentation

11.65.3.1 const unsigned int Parma_Polyhedra_Library::float_ieee754_single::BASE = 2 [static]

Definition at line 85 of file Float.defs.hh.

11.65.3.2 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::EXP_MASK = 0x7f800000 [static]

Definition at line 80 of file Float.defs.hh.

11.65.3.3 const int Parma_Polyhedra_Library::float_ieee754_single::EXPONENT_BIAS = EXPONENT_MAX [static]

Definition at line 89 of file Float.defs.hh.

Referenced by build().

11.65.3.4 const unsigned int Parma_Polyhedra_Library::float_ieee754_single::EXPONENT_BITS = 8 [static]

Definition at line 86 of file Float.defs.hh.

11.65.3.5 const int Parma_Polyhedra_Library::float_ieee754_single::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 88 of file Float.defs.hh.

11.65.3.6 const int Parma_Polyhedra_Library::float_ieee754_single::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 90 of file Float.defs.hh.

11.65.3.7 const int Parma_Polyhedra_Library::float_ieee754_single::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 91 of file Float.defs.hh.

11.65.3.8 const unsigned int Parma_Polyhedra_Library::float_ieee754_single::MANTISSA_BITS = 23 [static]

Definition at line 87 of file Float.defs.hh.

Referenced by build().

11.65.3.9 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::NEG_INF = 0xff800000 [static]

Definition at line 82 of file Float.defs.hh.

Referenced by is_inf().

11.65.3.10 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::NEG_ZERO = 0x80000000 [static]

Definition at line 84 of file Float.defs.hh.

Referenced by is_zero().

11.65.3.11 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::POS_INF = 0x7f800000 [static]

Definition at line 81 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.65.3.12 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::POS_ZERO = 0x00000000 [static]

Definition at line 83 of file Float.defs.hh.

Referenced by is_zero().

11.65.3.13 const uint32_t Parma_Polyhedra_Library::float_ieee754_single::SGN_MASK = 0x80000000 [static]

Definition at line 79 of file Float.defs.hh.

Referenced by build(), is_nan(), negate(), set_max(), and sign_bit().

11.65.3.14 uint32_t Parma_Polyhedra_Library::float_ieee754_single::word

Definition at line 78 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh

11.66 Parma_Polyhedra_Library::float_intel_double_extended Struct Reference

`#include <Float.defs.hh>`

Public Member Functions

- int is_inf () const
- int is_nan () const
- int is_zero () const
- int sign_bit () const
- void negate ()
- void dec ()
- void inc ()
- void set_max (bool negative)
- void build (bool negative, mpz_t mantissa, int exponent)

Public Attributes

- uint64_t lsp
- uint32_t msp

Static Public Attributes

- static const uint32_t MSP_SGN_MASK = 0x00008000
- static const uint32_t MSP_POS_INF = 0x00007fff
- static const uint32_t MSP_NEG_INF = 0x0000ffff
- static const uint32_t MSP_POS_ZERO = 0x00000000
- static const uint32_t MSP_NEG_ZERO = 0x00008000
- static const uint64_t LSP_INF = 0x8000000000000000ULL
- static const uint64_t LSP_ZERO = 0
- static const uint64_t LSP_DMAX = 0x7fffffffffffffffULL
- static const uint64_t LSP_NMAX = 0xffffffffffffffffULL
- static const unsigned int BASE = 2
- static const unsigned int EXPONENT_BITS = 15
- static const unsigned int MANTISSA_BITS = 63
- static const int EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1
- static const int EXPONENT_BIAS = EXPONENT_MAX
- static const int EXPONENT_MIN = -EXPONENT_MAX + 1
- static const int EXPONENT_MIN_DENORM

11.66.1 Detailed Description

Definition at line 195 of file Float.defs.hh.

11.66.2 Member Function Documentation

11.66.2.1 void Parma_Polyhedra_Library::float_intel_double_extended::build (bool *negative*, mpz_t *mantissa*, int *exponent*) [inline]

Definition at line 358 of file Float.inlines.hh.

References EXPONENT_BIAS, lsp, msp, and MSP_SGN_MASK.

11.66.2.2 void Parma_Polyhedra_Library::float_intel_double_extended::dec () [inline]

Definition at line 330 of file Float.inlines.hh.

References lsp, LSP_DMAX, LSP_NMAX, msp, and MSP_NEG_INF.

11.66.2.3 void Parma_Polyhedra_Library::float_intel_double_extended::inc () [inline]

Definition at line 340 of file Float.inlines.hh.

References lsp, LSP_DMAX, and msp.

11.66.2.4 int Parma_Polyhedra_Library::float_intel_double_extended::is_inf () const [inline]

Definition at line 290 of file Float.inlines.hh.

References lsp, LSP_INF, msp, MSP_NEG_INF, and MSP_POS_INF.

11.66.2.5 int Parma_Polyhedra_Library::float_intel_double_extended::is_nan () const [inline]

Definition at line 302 of file Float.inlines.hh.

References lsp, LSP_INF, msp, and MSP_POS_INF.

11.66.2.6 int Parma_Polyhedra_Library::float_intel_double_extended::is_zero () const [inline]

Definition at line 308 of file Float.inlines.hh.

References lsp, LSP_ZERO, msp, MSP_NEG_INF, MSP_NEG_ZERO, and MSP_POS_ZERO.

11.66.2.7 void Parma_Polyhedra_Library::float_intel_double_extended::negate () [inline]

Definition at line 320 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.66.2.8 void Parma_Polyhedra_Library::float_intel_double_extended::set_max (bool *negative*) [inline]

Definition at line 350 of file Float.inlines.hh.

References lsp, msp, and MSP_SGN_MASK.

11.66.2.9 int Parma_Polyhedra_Library::float_intel_double_extended::sign_bit () const [inline]

Definition at line 325 of file Float.inlines.hh.

References msp, and MSP_SGN_MASK.

11.66.3 Member Data Documentation

11.66.3.1 const unsigned int Parma_Polyhedra_Library::float_intel_double_extended::BASE = 2 [static]

Definition at line 212 of file Float.defs.hh.

11.66.3.2 const int Parma_Polyhedra_Library::float_intel_double_extended::EXPONENT_BIAS = EXPONENT_MAX [static]

Definition at line 216 of file Float.defs.hh.

Referenced by build().

11.66.3.3 const unsigned int Parma_Polyhedra_Library::float_intel_double_-extended::EXPONENT_BITS = 15 [static]

Definition at line 213 of file Float.defs.hh.

11.66.3.4 const int Parma_Polyhedra_Library::float_intel_double_extended::EXPONENT_MAX = (1 << (EXPONENT_BITS - 1)) - 1 [static]

Definition at line 215 of file Float.defs.hh.

11.66.3.5 const int Parma_Polyhedra_Library::float_intel_double_extended::EXPONENT_MIN = -EXPONENT_MAX + 1 [static]

Definition at line 217 of file Float.defs.hh.

11.66.3.6 const int Parma_Polyhedra_Library::float_intel_double_extended::EXPONENT_MIN_DENORM [static]

Initial value:

```
EXPONENT_MIN
                                        - static_cast<int>(MANTISSA_BITS)
```

Definition at line 218 of file Float.defs.hh.

11.66.3.7 uint64_t Parma_Polyhedra_Library::float_intel_double_extended::lsp

Definition at line 200 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), and set_max().

11.66.3.8 const uint64_t Parma_Polyhedra_Library::float_intel_double_extended::LSP_DMAX = 0x7fffffffffffffffULL [static]

Definition at line 210 of file Float.defs.hh.

Referenced by dec(), and inc().

11.66.3.9 const uint64_t Parma_Polyhedra_Library::float_intel_double_extended::LSP_INF = 0x8000000000000000ULL [static]

Definition at line 208 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.66.3.10 const uint64_t Parma_Polyhedra_Library::float_intel_double_extended::LSP_NMAX = 0xffffffffffffffffULL [static]

Definition at line 211 of file Float.defs.hh.

Referenced by dec().

11.66.3.11 const uint64_t Parma_Polyhedra_Library::float_intel_double_extended::LSP_ZERO = 0 [static]

Definition at line 209 of file Float.defs.hh.

Referenced by is_zero().

11.66.3.12 const unsigned int Parma_Polyhedra_Library::float_intel_double_extended::MANTISSA_BITS = 63 [static]

Definition at line 214 of file Float.defs.hh.

11.66.3.13 uint32_t Parma_Polyhedra_Library::float_intel_double_extended::msp

Definition at line 201 of file Float.defs.hh.

Referenced by build(), dec(), inc(), is_inf(), is_nan(), is_zero(), negate(), set_max(), and sign_bit().

11.66.3.14 const uint32_t Parma_Polyhedra_Library::float_intel_double_extended::MSP_NEG_INF = 0x0000ffff [static]

Definition at line 205 of file Float.defs.hh.

Referenced by dec(), is_inf(), and is_zero().

11.66.3.15 const uint32_t Parma_Polyhedra_Library::float_intel_double_extended::MSP_NEG_ZERO = 0x00008000 [static]

Definition at line 207 of file Float.defs.hh.

Referenced by is_zero().

11.66.3.16 const uint32_t Parma_Polyhedra_Library::float_intel_double_extended::MSP_POS_INF = 0x00007fff [static]

Definition at line 204 of file Float.defs.hh.

Referenced by is_inf(), and is_nan().

11.66.3.17 const uint32_t Parma_Polyhedra_Library::float_intel_double_extended::MSP_POS_ZERO = 0x00000000 [static]

Definition at line 206 of file Float.defs.hh.

Referenced by is_zero().

11.66.3.18 const uint32_t Parma_Polyhedra_Library::float_intel_double_extended::MSP_SGN_MASK = 0x00008000 [static]

Definition at line 203 of file Float.defs.hh.

Referenced by build(), negate(), set_max(), and sign_bit().

The documentation for this struct was generated from the following files:

- Float.defs.hh
- Float.inlines.hh
- Float.cc

11.67 Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > > Struct Template Reference

`#include <Checked_Number.defs.hh>`

11.67.1 Detailed Description

template<typename T, typename Policy> struct Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > >

Definition at line 1036 of file Checked_Number.defs.hh.

The documentation for this struct was generated from the following file:

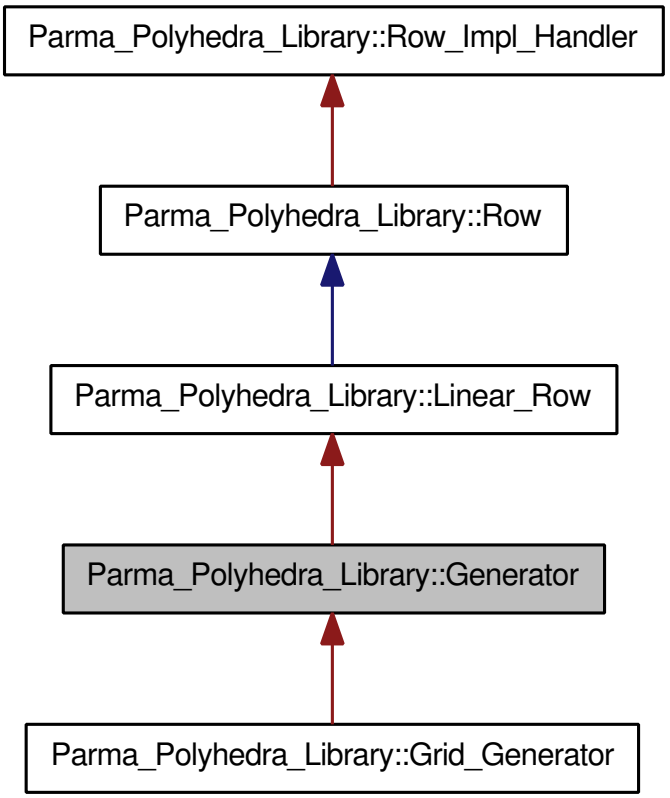
- Checked_Number.defs.hh

11.68 Parma_Polyhedra_Library::Generator Class Reference

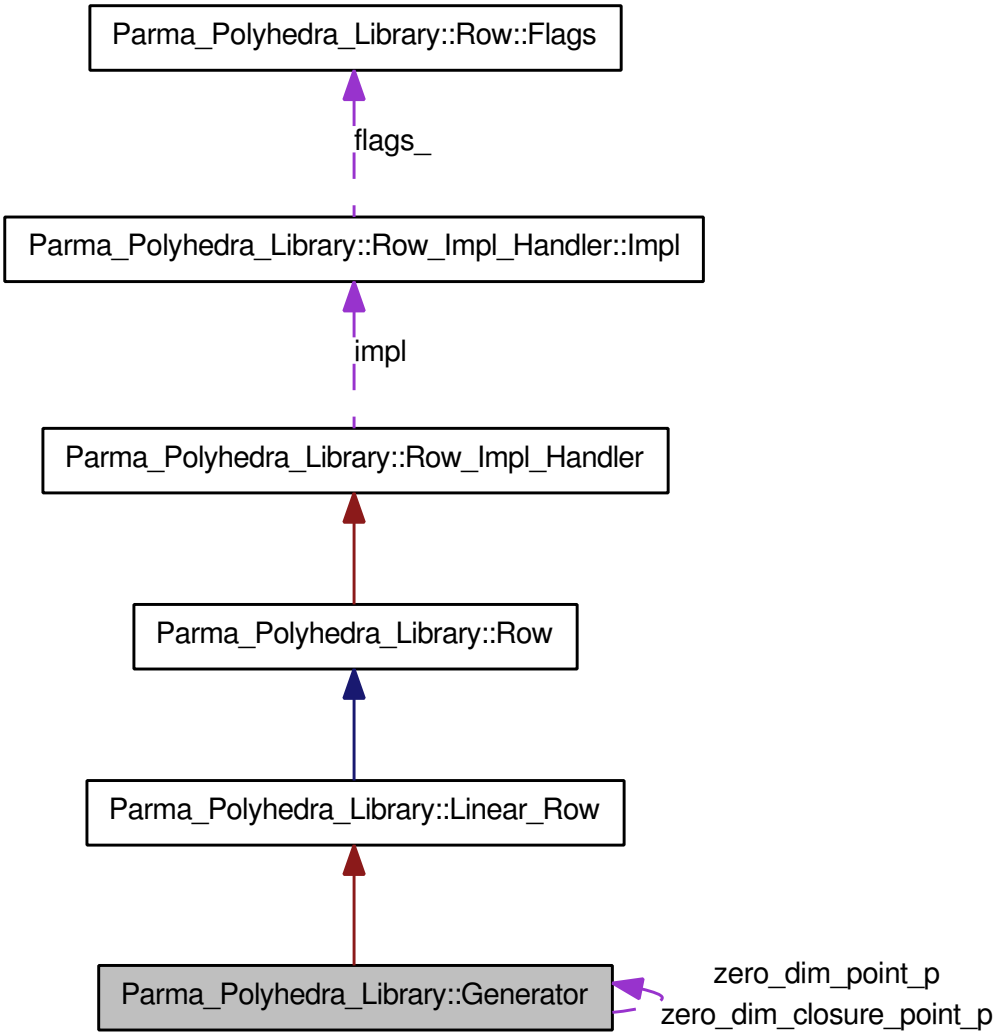
A line, ray, point or closure point.

`#include <Generator.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Generator:



Collaboration diagram for Parma_Polyhedra_Library::Generator:



Public Types

- enum Type { LINE, RAY, POINT, CLOSURE_POINT }

  The generator type.

Public Member Functions

- Generator (const Generator &g)

  Ordinary copy constructor.

- ~Generator ()

  Destructor.

- Generator & operator= (const Generator &g)

Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Type type () const

  Returns the generator type of `*this`.

- bool is_line () const

  Returns `true` *if and only if* `*this` *is a line.*

- bool is_ray () const

  Returns `true` *if and only if* `*this` *is a ray.*

- bool is_line_or_ray () const

  Returns `true` *if and only if* `*this` *is a line or a ray.*

- bool is_point () const

  Returns `true` *if and only if* `*this` *is a point.*

- bool is_closure_point () const

  Returns `true` *if and only if* `*this` *is a closure point.*

- Coefficient_traits::const_reference coefficient (Variable v) const

  Returns the coefficient of `v` *in* `*this`.

- Coefficient_traits::const_reference divisor () const

  If `*this` *is either a point or a closure point, returns its divisor.*

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool is_equivalent_to (const Generator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are equivalent generators.*

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void swap (Generator &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static Generator line (const Linear_Expression &e)

  Returns the line of direction `e`.

- static Generator ray (const Linear_Expression &e)

  Returns the ray of direction `e`.

- static Generator point (const Linear_Expression &e=Linear_Expression::zero(), Coefficient_traits::const_reference d=Coefficient_one())

  Returns the point at `e / d`.

- static Generator closure_point (const Linear_Expression &e=Linear_Expression::zero(), Coefficient_traits::const_reference d=Coefficient_one())

  Returns the closure point at `e / d`.

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Generator can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Generator & zero_dim_point ()

  Returns the origin of the zero-dimensional space $\mathbb{R}^0$.

- static const Generator & zero_dim_closure_point ()

  Returns, as a closure point, the origin of the zero-dimensional space $\mathbb{R}^0$.

Private Member Functions

- Generator (Linear_Expression &e, Type type, Topology topology)

  Builds a generator of type `type` *and topology* `topology`*, stealing the coefficients from* `e`.

- void throw_dimension_incompatible (const char *method, const char *name_var, Variable v) const

  Throw a `std::invalid_argument` *exception containing the appropriate error message.*

- void throw_invalid_argument (const char *method, const char *reason) const

Throw a `std::invalid_argument` exception containing the appropriate error message.

- friend Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Generator &g)
- Generator (const Generator &g, dimension_type dimension)

  Copy constructor with given space dimension.

- bool is_ray_or_point () const

  Returns `true` if and only if `*this` is not a line.

- void set_is_line ()

  Sets the Linear_Row kind to `LINE_OR_EQUALITY`.

- void set_is_ray_or_point ()

  Sets the Linear_Row kind to `RAY_OR_POINT_OR_INEQUALITY`.

- bool is_matching_closure_point (const Generator &p) const

  Returns `true` if and only if the closure point `*this` has the same coordinates of the point `p`.

- Generator ()

  Default constructor: private and not implemented.

Static Private Attributes

- static const Generator * zero_dim_point_p = 0

  Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$.

- static const Generator * zero_dim_closure_point_p = 0

  Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$, as a closure point.

Friends

- class Parma_Polyhedra_Library::Scalar_Products
- class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign
- class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Assign
- class Parma_Polyhedra_Library::Generator_System
- class Parma_Polyhedra_Library::Generator_System::const_iterator
- class Parma_Polyhedra_Library::Polyhedron
- class Parma_Polyhedra_Library::Grid_Generator
- class Parma_Polyhedra_Library::Grid_Generator_System
- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Generator &g)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Generator &g)

  Output operator.

- void swap (Parma_Polyhedra_Library::Generator &x, Parma_Polyhedra_Library::Generator &y)

  Specializes `std::swap`.

- bool operator== (const Generator &x, const Generator &y)

  Returns `true` *if and only if* `x` *is equivalent to* `y`.

- bool operator!= (const Generator &x, const Generator &y)

  Returns `true` *if and only if* `x` *is not equivalent to* `y`.

- template<typename To >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`.

- template<typename Temp , typename To >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`.

- template<typename To >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir)

  Computes the euclidean distance between `x` *and* `y`.

- template<typename Temp , typename To >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the euclidean distance between `x` *and* `y`.

- template<typename To >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir)

  Computes the L_∞ *distance between* `x` *and* `y`.

- template<typename Temp , typename To >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ *distance between* `x` *and* `y`.

- std::ostream & operator<< (std::ostream &s, const Generator::Type &t)

  Output operator.

- template<typename Specialization , typename Temp , typename To >
  bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Generator &x, const Generator &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

11.68.1 Detailed Description

A line, ray, point or closure point. An object of the class Generator is one of the following:

- a line $\boldsymbol{l} = (a_0, \ldots, a_{n-1})^{\mathrm{T}}$;
- a ray $\boldsymbol{r} = (a_0, \ldots, a_{n-1})^{\mathrm{T}}$;
- a point $\boldsymbol{p} = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$;
- a closure point $\boldsymbol{c} = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$;

where n is the dimension of the space and, for points and closure points, $d > 0$ is the divisor.

A note on terminology.

As observed in Section Representations of Convex Polyhedra, there are cases when, in order to represent a polyhedron $\mathcal{P}$ using the generator system $\mathcal{G} = (L, R, P, C)$, we need to include in the finite set P even points of $\mathcal{P}$ that are *not* vertices of $\mathcal{P}$. This situation is even more frequent when working with NNC polyhedra and it is the reason why we prefer to use the word 'point' where other libraries use the word 'vertex'.

How to build a generator.

Each type of generator is built by applying the corresponding function (`line`, `ray`, `point` or `closure_point`) to a linear expression, representing a direction in the space; the space dimension of the generator is defined as the space dimension of the corresponding linear expression. Linear expressions used to define a generator should be homogeneous (any constant term will be simply ignored). When defining points and closure points, an optional Coefficient argument can be used as a common *divisor* for all the coefficients occurring in the provided linear expression; the default value for this argument is 1.

In all the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds a line with direction $x - y - z$ and having space dimension 3:

```
Generator l = line(x - y - z);
```

As mentioned above, the constant term of the linear expression is not relevant. Thus, the following code has the same effect:

```
Generator l = line(x - y - z + 15);
```

By definition, the origin of the space is not a line, so that the following code throws an exception:

```
Generator l = line(0*x);
```

Example 2

The following code builds a ray with the same direction as the line in Example 1:

```
Generator r = ray(x - y - z);
```

As is the case for lines, when specifying a ray the constant term of the linear expression is not relevant; also, an exception is thrown when trying to build a ray from the origin of the space.

Example 3

The following code builds the point $\boldsymbol{p} = (1, 0, 2)^{\mathrm{T}} \in \mathbb{R}^3$:

```
Generator p = point(1*x + 0*y + 2*z);
```

The same effect can be obtained by using the following code:

```
Generator p = point(x + 2*z);
```

Similarly, the origin $\mathbf{0} \in \mathbb{R}^3$ can be defined using either one of the following lines of code:

```
Generator origin3 = point(0*x + 0*y + 0*z);
Generator origin3_alt = point(0*z);
```

Note however that the following code would have defined a different point, namely $\mathbf{0} \in \mathbb{R}^2$:

```
Generator origin2 = point(0*y);
```

The following two lines of code both define the only point having space dimension zero, namely $\mathbf{0} \in \mathbb{R}^0$. In the second case we exploit the fact that the first argument of the function `point` is optional.

```
Generator origin0 = Generator::zero_dim_point();
Generator origin0_alt = point();
```

Example 4

The point $\boldsymbol{p}$ specified in Example 3 above can also be obtained with the following code, where we provide a non-default value for the second argument of the function `point` (the divisor):

```
Generator p = point(2*x + 0*y + 4*z, 2);
```

Obviously, the divisor can be usefully exploited to specify points having some non-integer (but rational) coordinates. For instance, the point $\boldsymbol{q} = (-1.5, 3.2, 2.1)^{\mathrm{T}} \in \mathbb{R}^3$ can be specified by the following code:

```
Generator q = point(-15*x + 32*y + 21*z, 10);
```

If a zero divisor is provided, an exception is thrown.

Example 5

Closure points are specified in the same way we defined points, but invoking their specific constructor function. For instance, the closure point $\boldsymbol{c} = (1, 0, 2)^{\mathrm{T}} \in \mathbb{R}^3$ is defined by

```
Generator c = closure_point(1*x + 0*y + 2*z);
```

For the particular case of the (only) closure point having space dimension zero, we can use any of the following:

```
Generator closure_origin0 = Generator::zero_dim_closure_point();
Generator closure_origin0_alt = closure_point();
```

How to inspect a generator

Several methods are provided to examine a generator and extract all the encoded information: its space dimension, its type and the value of its integer coefficients.

Example 6

The following code shows how it is possible to access each single coefficient of a generator. If `g1` is a point having coordinates $(a_0, \ldots, a_{n-1})^{\mathrm{T}}$, we construct the closure point `g2` having coordinates $(a_0, 2a_1, \ldots, (i+1)a_i, \ldots, na_{n-1})^{\mathrm{T}}$.

```
if (g1.is_point()) {
  cout << "Point g1: " << g1 << endl;
  Linear_Expression e;
  for (dimension_type i = g1.space_dimension(); i-- > 0; )
    e += (i + 1) * g1.coefficient(Variable(i)) * Variable(i);
  Generator g2 = closure_point(e, g1.divisor());
  cout << "Closure point g2: " << g2 << endl;
}
else
  cout << "Generator g1 is not a point." << endl;
```

Therefore, for the point

```
Generator g1 = point(2*x - y + 3*z, 2);
```

we would obtain the following output:

```
Point g1: p((2*A - B + 3*C)/2)
Closure point g2: cp((2*A - 2*B + 9*C)/2)
```

When working with (closure) points, be careful not to confuse the notion of *coefficient* with the notion of *coordinate*: these are equivalent only when the divisor of the (closure) point is 1.

Definition at line 244 of file Generator.defs.hh.

11.68.2 Member Enumeration Documentation

11.68.2.1 enum Parma_Polyhedra_Library::Generator::Type

The generator type.

Enumerator:

LINE The generator is a line.

RAY The generator is a ray.

POINT The generator is a point.

CLOSURE_POINT The generator is a closure point.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 303 of file Generator.defs.hh.

11.68.3 Constructor & Destructor Documentation

11.68.3.1 Parma_Polyhedra_Library::Generator::Generator (const Generator & *g*) [inline]

Ordinary copy constructor.

Definition at line 39 of file Generator.inlines.hh.

11.68.3.2 Parma_Polyhedra_Library::Generator::~Generator () [inline]

Destructor.

Definition at line 49 of file Generator.inlines.hh.

11.68.3.3 Parma_Polyhedra_Library::Generator::Generator (Linear_Expression & *e*, Type *type*, Topology *topology*) [inline, private]

Builds a generator of type `type` and topology `topology`, stealing the coefficients from `e`.

Definition at line 30 of file Generator.inlines.hh.

References CLOSURE_POINT, Parma_Polyhedra_Library::Linear_Row::flags(), LINE, Parma_Polyhedra_Library::Linear_Row::LINE_OR_EQUALITY, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, and swap().

11.68.3.4 Parma_Polyhedra_Library::Generator::Generator (const Generator & *g*, dimension_type *dimension*) [inline, private]

Copy constructor with given space dimension.

Definition at line 44 of file Generator.inlines.hh.

11.68.3.5 Parma_Polyhedra_Library::Generator::Generator () [private]

Default constructor: private and not implemented.

Referenced by initialize().

11.68.4 Member Function Documentation

11.68.4.1 void Parma_Polyhedra_Library::Generator::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 199 of file Generator.inlines.hh.

References ascii_dump().

11.68.4.2 void Parma_Polyhedra_Library::Generator::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Referenced by Parma_Polyhedra_Library::MIP_Problem::ascii_dump(), and ascii_dump().

11.68.4.3 bool Parma_Polyhedra_Library::Generator::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 204 of file Generator.inlines.hh.

11.68.4.4 PPL::Generator Parma_Polyhedra_Library::Generator::closure_point (const Linear_Expression & *e* = `Linear_Expression::zero()`, Coefficient_traits::const_reference *d* = `Coefficient_one()`) [inline, static]

Returns the closure point at `e` / `d`.

Shorthand for Generator Generator::closure_point(const Linear_Expression& e, Coefficient_traits::const_reference d).

Both `e` and `d` are optional arguments, with default values Linear_Expression::zero() and Coefficient_one(), respectively.

Exceptions

std::invalid_argument Thrown if `d` is zero.

Definition at line 181 of file Generator.inlines.hh.

Referenced by initialize().

11.68.4.5 Coefficient_traits::const_reference Parma_Polyhedra_Library::Generator::coefficient (Variable *v*) const [inline]

Returns the coefficient of `v` in `*this`.

Exceptions

std::invalid_argument Thrown if the index of `v` is greater than or equal to the space dimension of `*this`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 124 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Variable::id(), space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::MIP_Problem::choose_branching_variable(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable(), l_m_-

distance_assign(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::MIP_-Problem::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_-Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), and Parma_Polyhedra_Library::MIP_-Problem::solve_mip().

11.68.4.6 Coefficient_traits::const_reference Parma_Polyhedra_Library::Generator::divisor () const [inline]

If `*this` is either a point or a closure point, returns its divisor.

Exceptions

std::invalid_argument Thrown if `*this` is neither a point nor a closure point.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 131 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::inhomogeneous_term(), is_ray_or_point(), and throw_invalid_argument().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_-Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_-Polyhedra_Library::MIP_Problem::choose_branching_variable(), Parma_Polyhedra_Library::MIP_-Problem::evaluate_objective_function(), Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable(), l_m_distance_assign(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::MIP_-Problem::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_-Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), and Parma_Polyhedra_Library::MIP_-Problem::solve_mip().

11.68.4.7 memory_size_type Parma_Polyhedra_Library::Generator::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 140 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::external_memory_in_bytes().

11.68.4.8 void Parma_Polyhedra_Library::Generator::finalize () [static]

Finalizes the class.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 176 of file Generator.cc.

References zero_dim_closure_point_p, and zero_dim_point_p.

11.68.4.9 void Parma_Polyhedra_Library::Generator::initialize () [static]

Initializes the class.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 165 of file Generator.cc.

References closure_point(), Generator(), point(), zero_dim_closure_point_p, and zero_dim_point_p.

11.68.4.10 bool Parma_Polyhedra_Library::Generator::is_closure_point () const [inline]

Returns `true` if and only if `*this` is a closure point.

Definition at line 109 of file Generator.inlines.hh.

References CLOSURE_POINT, and type().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Grid::relation_with(), and Parma_Polyhedra_Library::Generator_System::const_iterator::skip_forward().

11.68.4.11 bool Parma_Polyhedra_Library::Generator::is_equivalent_to (const Generator & *y*) const

Returns `true` if and only if `*this` and `y` are equivalent generators.

Generators having different space dimensions are not equivalent.

Definition at line 126 of file Generator.cc.

References Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), Parma_Polyhedra_Library::Row::normalize(), POINT, space_dimension(), and type().

Referenced by operator!=(), and operator==().

11.68.4.12 bool Parma_Polyhedra_Library::Generator::is_line () const [inline]

Returns `true` if and only if `*this` is a line.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 69 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Polyhedron::bounds(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and type().

11.68.4.13 bool Parma_Polyhedra_Library::Generator::is_line_or_ray () const [inline]

Returns `true` if and only if `*this` is a line or a ray.

Definition at line 79 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Generator_System::add_corresponding_points(), Parma_Polyhedra_Library::Polyhedron::bounds(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter(), is_ray(), l_m_distance_assign(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), and type().

11.68.4.14 bool Parma_Polyhedra_Library::Generator::is_matching_closure_point (const Generator & *p*) const [private]

Returns `true` if and only if the closure point `*this` has the same *coordinates* of the point `p`.

It is *assumed* that `*this` is a closure point, `p` is a point and both topologies and space dimensions agree.

Definition at line 276 of file Generator.cc.

References CLOSURE_POINT, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), POINT, PPL_DIRTY_TEMP_COEFFICIENT, space_dimension(), Parma_Polyhedra_Library::Linear_Row::topology(), and type().

Referenced by Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), and Parma_Polyhedra_Library::Generator_System::const_iterator::skip_forward().

11.68.4.15 bool Parma_Polyhedra_Library::Generator::is_point () const [inline]

Returns `true` if and only if `*this` is a point.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 104 of file Generator.inlines.hh.

References POINT, and type().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::Generator_System::const_iterator::skip_forward().

11.68.4.16 bool Parma_Polyhedra_Library::Generator::is_ray () const [inline]

Returns `true` if and only if `*this` is a ray.

Definition at line 84 of file Generator.inlines.hh.

References is_line_or_ray(), and is_ray_or_point().

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Grid_Generator::is_parameter(), and Parma_Polyhedra_Library::Box< ITV >::relation_with().

11.68.4.17 bool Parma_Polyhedra_Library::Generator::is_ray_or_point () const [inline, private]

Returns `true` if and only if `*this` is not a line.

Definition at line 74 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality().

Referenced by divisor(), and is_ray().

11.68.4.18 PPL::Generator Parma_Polyhedra_Library::Generator::line (const Linear_Expression & *e*) [inline, static]

Returns the line of direction `e`.

Shorthand for Generator Generator::line(const Linear_Expression& e).

Exceptions

std::invalid_argument Thrown if the homogeneous part of `e` represents the origin of the vector space.

Definition at line 163 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), and Parma_Polyhedra_Library::Polyhedron::unconstrain().

11.68.4.19 dimension_type Parma_Polyhedra_Library::Generator::max_space_dimension () [inline, static]

Returns the maximum space dimension a Generator can handle.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 59 of file Generator.inlines.hh.

11.68.4.20 bool Parma_Polyhedra_Library::Generator::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 319 of file Generator.cc.

References Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_Row::OK(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), and type().

Referenced by Parma_Polyhedra_Library::MIP_Problem::OK().

11.68.4.21 Generator & Parma_Polyhedra_Library::Generator::operator= (const Generator & *g*) [inline]

Assignment operator.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 53 of file Generator.inlines.hh.

11.68.4.22 Parma_Polyhedra_Library::Generator::Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Generator & *g*) [private]

11.68.4.23 PPL::Generator Parma_Polyhedra_Library::Generator::point (const Linear_Expression & *e* = `Linear_Expression::zero()`, Coefficient_traits::const_reference *d* = `Coefficient_one()`) [inline, static]

Returns the point at `e` / `d`.

Shorthand for Generator Generator::point(const Linear_Expression& e, Coefficient_traits::const_reference d).

Both e and d are optional arguments, with default values Linear_Expression::zero() and Coefficient_one(), respectively.

Exceptions

std::invalid_argument Thrown if d is zero.

Definition at line 175 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), initialize(), Parma_Polyhedra_Library::Grid::max_min(), and Parma_Polyhedra_Library::Box< ITV >::max_min().

11.68.4.24 void Parma_Polyhedra_Library::Generator::print () const

Prints *this to std::cerr using operator<<.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

11.68.4.25 PPL::Generator Parma_Polyhedra_Library::Generator::ray (const Linear_Expression & *e*) [inline, static]

Returns the ray of direction e.

Shorthand for Generator Generator::ray(const Linear_Expression& e).

Exceptions

std::invalid_argument Thrown if the homogeneous part of e represents the origin of the vector space.

Definition at line 169 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator().

11.68.4.26 void Parma_Polyhedra_Library::Generator::set_is_line () [inline, private]

Sets the Linear_Row kind to LINE_OR_EQUALITY.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 114 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators().

11.68.4.27 void Parma_Polyhedra_Library::Generator::set_is_ray_or_point () [inline, private]

Sets the Linear_Row kind to `RAY_OR_POINT_OR_INEQUALITY`.

Definition at line 119 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::set_is_ray_or_point_or_inequality().

Referenced by Parma_Polyhedra_Library::Grid_Generator::coefficient_swap(), and Parma_Polyhedra_Library::Grid_Generator::set_is_parameter_or_point().

11.68.4.28 dimension_type Parma_Polyhedra_Library::Generator::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 64 of file Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), coefficient(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), is_equivalent_to(), is_matching_closure_point(), Parma_Polyhedra_Library::MIP_Problem::is_satisfied(), Parma_Polyhedra_Library::MIP_Problem::is_saturated(), l_m_distance_assign(), Parma_Polyhedra_Library::MIP_Problem::OK(), Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator()(), operator<<(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.68.4.29 void Parma_Polyhedra_Library::Generator::swap (Generator & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 209 of file Generator.inlines.hh.

Referenced by Generator(), and swap().

11.68.4.30 void Parma_Polyhedra_Library::Generator::throw_dimension_incompatible (const char * *method*, const char * *name_var*, Variable *v*) const [private]

Throw a `std::invalid_argument` exception containing the appropriate error message.

Definition at line 36 of file Generator.cc.

References Parma_Polyhedra_Library::Variable::space_dimension(), and space_dimension().

Referenced by Parma_Polyhedra_Library::Grid_Generator::coefficient(), and coefficient().

11.68.4.31 void Parma_Polyhedra_Library::Generator::throw_invalid_argument (const char * *method*, const char * *reason*) const [private]

Throw a `std::invalid_argument` exception containing the appropriate error message.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 47 of file Generator.cc.

Referenced by divisor().

11.68.4.32 memory_size_type Parma_Polyhedra_Library::Generator::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 145 of file Generator.inlines.hh.

11.68.4.33 Generator::Type Parma_Polyhedra_Library::Generator::type () const [inline]

Returns the generator type of `*this`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 89 of file Generator.inlines.hh.

References CLOSURE_POINT, is_line(), is_line_or_ray(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), LINE, POINT, RAY, and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), is_closure_point(), is_equivalent_to(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), is_matching_closure_point(), is_point(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), operator<<(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_Library::Grid_Generator::type().

11.68.4.34 const Generator & Parma_Polyhedra_Library::Generator::zero_dim_closure_point () [inline, static]

Returns, as a closure point, the origin of the zero-dimensional space $\mathbb{R}^0$.

Definition at line 156 of file Generator.inlines.hh.

References zero_dim_closure_point_p.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project().

11.68.4.35 const Generator & Parma_Polyhedra_Library::Generator::zero_dim_point () [inline, static]

Returns the origin of the zero-dimensional space $\mathbb{R}^0$.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 150 of file Generator.inlines.hh.

References zero_dim_point_p.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), and Parma_Polyhedra_Library::Generator_System::initialize().

11.68.5 Friends And Related Function Documentation

11.68.5.1 template<typename Temp , typename To > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 314 of file Generator.inlines.hh.

11.68.5.2 template<typename To > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 328 of file Generator.inlines.hh.

11.68.5.3 template<typename Temp , typename To > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 352 of file Generator.inlines.hh.

11.68.5.4 template<typename To > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 366 of file Generator.inlines.hh.

11.68.5.5 template<typename Specialization , typename Temp , typename To > bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Definition at line 218 of file Generator.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), coefficient(), Parma_Polyhedra_Library::combine(), divisor(), Parma_Polyhedra_Library::finalize(), is_line_or_ray(), Parma_Polyhedra_Library::maybe_assign(), and space_dimension().

11.68.5.6 bool operator!= (const Generator & *x*, const Generator & *y*) [related]

Returns `true` if and only if `x` is not equivalent to `y`.

Definition at line 194 of file Generator.inlines.hh.

References is_equivalent_to().

11.68.5.7 std::ostream & operator<< (std::ostream & *s*, const Generator::Type & *t*) [related]

Output operator.

Definition at line 255 of file Generator.cc.

References CLOSURE_POINT, LINE, POINT, and RAY.

11.68.5.8 std::ostream & operator<< (std::ostream & *s*, const Generator & *g*) [related]

Output operator.

Definition at line 188 of file Generator.cc.

References CLOSURE_POINT, LINE, Parma_Polyhedra_Library::neg_assign(), POINT, PPL_DIRTY_TEMP_COEFFICIENT, RAY, space_dimension(), and type().

11.68.5.9 bool operator== (const Generator & *x*, const Generator & *y*) [related]

Returns `true` if and only if `x` is equivalent to `y`.

Definition at line 188 of file Generator.inlines.hh.

References is_equivalent_to().

11.68.5.10 friend class Parma_Polyhedra_Library::Generator_System [friend]

Definition at line 434 of file Generator.defs.hh.

11.68.5.11 friend class Parma_Polyhedra_Library::Generator_System::const_iterator [friend]

Definition at line 435 of file Generator.defs.hh.

11.68.5.12 friend class Parma_Polyhedra_Library::Grid_Generator [friend]

Definition at line 438 of file Generator.defs.hh.

11.68.5.13 friend class Parma_Polyhedra_Library::Grid_Generator_System [friend]

Definition at line 440 of file Generator.defs.hh.

11.68.5.14 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Generator & *g*) [friend]

11.68.5.15 friend class Parma_Polyhedra_Library::Polyhedron [friend]

Definition at line 437 of file Generator.defs.hh.

11.68.5.16 friend class Parma_Polyhedra_Library::Scalar_Products [friend]

Definition at line 431 of file Generator.defs.hh.

11.68.5.17 friend class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Assign [friend]

Definition at line 433 of file Generator.defs.hh.

11.68.5.18 friend class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign [friend]

Definition at line 432 of file Generator.defs.hh.

11.68.5.19 template<typename Temp , typename To > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 276 of file Generator.inlines.hh.

11.68.5.20 template<typename To > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Generator & *x*, const Generator & *y*, Rounding_Dir *dir*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

Computes the euclidean distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_-Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_-Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_-Policy>.

Note

Distances are *only* defined between generators that are points and/or closure points; for rays or lines, `false` is returned.

Definition at line 290 of file Generator.inlines.hh.

11.68.5.21 void swap (Parma_Polyhedra_Library::Generator & *x*, Parma_Polyhedra_Library::Generator & *y*) [related]

Specializes `std::swap`.

Definition at line 393 of file Generator.inlines.hh.

References swap().

11.68.6 Member Data Documentation

11.68.6.1 const PPL::Generator * Parma_Polyhedra_Library::Generator::zero_dim_closure_-point_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$, as a closure point.

Definition at line 407 of file Generator.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_closure_point().

11.68.6.2 const PPL::Generator * Parma_Polyhedra_Library::Generator::zero_dim_point_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 401 of file Generator.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_point().

The documentation for this class was generated from the following files:

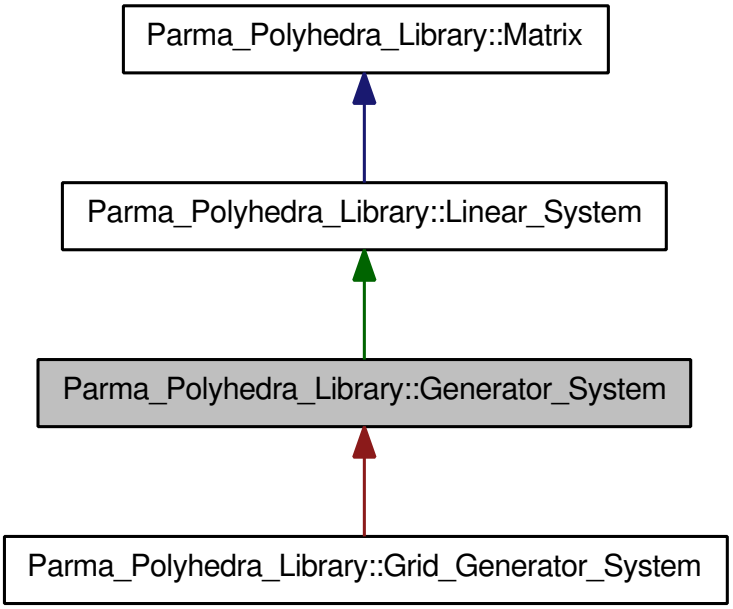
- Generator.defs.hh
- Generator.inlines.hh
- Generator.cc

11.69 Parma_Polyhedra_Library::Generator_System Class Reference

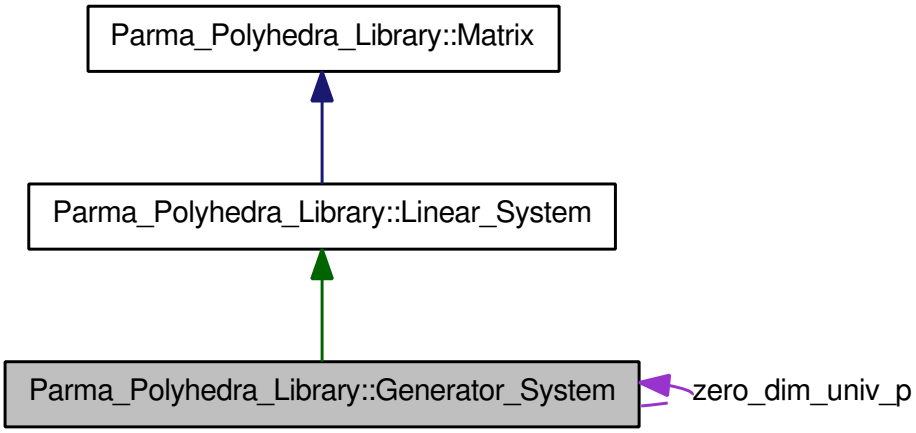
A system of generators.

`#include <Generator_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Generator_System:



Collaboration diagram for Parma_Polyhedra_Library::Generator_System:



Classes

- class const_iterator

  An iterator over a system of generators.

Public Member Functions

- Generator_System ()

  Default constructor: builds an empty system of generators.

- Generator_System (const Generator &g)

  Builds the singleton system containing only generator `g`.

- Generator_System (const Generator_System &gs)

  Ordinary copy constructor.

- ~Generator_System ()

  Destructor.

- Generator_System & operator= (const Generator_System &y)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- void clear ()

  Removes all the generators from the generator system and sets its space dimension to 0.

- void insert (const Generator &g)

  Inserts in `*this` *a copy of the generator* `g`, *increasing the number of space dimensions if needed.*

- bool empty () const

Returns `true` *if and only if* `*this` *has no generators.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first generator, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void swap (Generator_System &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Generator_System can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Generator_System & zero_dim_univ ()

  Returns the singleton system containing only Generator::zero_dim_point().

Private Member Functions

- Generator_System (Topology topol)

  Builds an empty system of generators having the specified topology.

- Generator_System (Topology topol, dimension_type n_rows, dimension_type n_columns)

  Builds a system of `n_rows` *rays/points on a* `n_columns` *- 1 dimensional space (including the* ϵ *dimension, if* `topol` *is* `NOT_NECESSARILY_CLOSED`*).*

- bool adjust_topology_and_space_dimension (Topology topol, dimension_type num_dimensions)

  Adjusts `*this` *so that it matches the topology and the number of space dimensions given as parameters (adding or removing columns if needed). Returns* `false` *if and only if* `topol` *is equal to* `NECESSARILY_CLOSED` *and* `*this` *contains closure points.*

- void add_corresponding_points ()

  For each unmatched closure point in `*this`*, adds the corresponding point.*

- bool has_points () const

  Returns `true` *if and only if* `*this` *contains one or more points.*

- void add_corresponding_closure_points ()

  For each unmatched point in `*this`*, adds the corresponding closure point.*

- bool has_closure_points () const

  Returns `true` *if and only if* `*this` *contains one or more closure points.*

- Generator & operator[ ] (dimension_type k)

  Returns the `k` *− th generator of the system.*

- const Generator & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k` *− th generator of the system.*

- Parma_Polyhedra_Library::Poly_Con_Relation relation_with (const Constraint &c) const

  Returns the relations holding between the generator system and the constraint `c`*.*

- bool satisfied_by_all_generators (const Constraint &c) const

  Returns `true` *if all the generators satisfy* `c`*.*

- bool satisfied_by_all_generators_C (const Constraint &c) const

  Returns `true` *if all the generators satisfy* `c`*.*

- bool satisfied_by_all_generators_NNC (const Constraint &c) const

  Returns `true` *if all the generators satisfy* `c`*.*

- void affine_image (dimension_type v, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator)

  Assigns to a given variable an affine expression.

- dimension_type num_lines () const

  Returns the number of lines of the system.

- dimension_type num_rays () const

  Returns the number of rays of the system.

- void remove_invalid_lines_and_rays ()

  Removes all the invalid lines and rays.

- void simplify ()

  Applies Gaussian elimination and back-substitution so as to provide a partial simplification of the system of generators.

- void insert_pending (const Generator &g)

  Inserts in `*this` *a copy of the generator* `g`*, increasing the number of space dimensions if needed. It is a pending generator.*

Static Private Attributes

- static const Generator_System * zero_dim_univ_p = 0

  Holds (between class initialization and finalization) a pointer to the singleton system containing only Generator::zero_dim_point().

Friends

- class const_iterator
- class Parma_Polyhedra_Library::Polyhedron
- class Parma_Polyhedra_Library::Grid_Generator_System
- bool operator== (const Polyhedron &x, const Polyhedron &y)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Generator_System &gs)

  Output operator.

- void swap (Parma_Polyhedra_Library::Generator_System &x, Parma_Polyhedra_Library::Generator_System &y)

  Specializes `std::swap`*.*

11.69.1 Detailed Description

A system of generators. An object of the class Generator_System is a system of generators, i.e., a multiset of objects of the class Generator (lines, rays, points and closure points). When inserting generators in a system, space dimensions are automatically adjusted so that all the generators in the system are defined on the same vector space. A system of generators which is meant to define a non-empty polyhedron must include at least one point: the reason is that lines, rays and closure points need a supporting point (lines and rays only specify directions while closure points only specify points in the topological closure of the NNC polyhedron).

In all the examples it is assumed that variables x and y are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code defines the line having the same direction as the x axis (i.e., the first Cartesian axis) in $\mathbb{R}^2$:

```
Generator_System gs;
gs.insert(line(x + 0*y));
```

As said above, this system of generators corresponds to an empty polyhedron, because the line has no supporting point. To define a system of generators that does correspond to the x axis, we can add the following code which inserts the origin of the space as a point:

```
gs.insert(point(0*x + 0*y));
```

Since space dimensions are automatically adjusted, the following code obtains the same effect:

```
gs.insert(point(0*x));
```

In contrast, if we had added the following code, we would have defined a line parallel to the x axis through the point $(0, 1)^{\mathrm{T}} \in \mathbb{R}^2$.

```
gs.insert(point(0*x + 1*y));
```

Example 2

The following code builds a ray having the same direction as the positive part of the x axis in $\mathbb{R}^2$:

```
Generator_System gs;
gs.insert(ray(x + 0*y));
```

To define a system of generators indeed corresponding to the set

$$\left\{ (x, 0)^{\mathrm{T}} \in \mathbb{R}^2 \;\middle|\; x \geq 0 \right\},$$

one just has to add the origin:

```
gs.insert(point(0*x + 0*y));
```

Example 3

The following code builds a system of generators having four points and corresponding to a square in $\mathbb{R}^2$ (the same as Example 1 for the system of constraints):

```
Generator_System gs;
gs.insert(point(0*x + 0*y));
gs.insert(point(0*x + 3*y));
gs.insert(point(3*x + 0*y));
gs.insert(point(3*x + 3*y));
```

Example 4

By using closure points, we can define the *kernel* (i.e., the largest open set included in a given set) of the square defined in the previous example. Note that a supporting point is needed and, for that purpose, any inner point could be considered.

```
Generator_System gs;
gs.insert(point(x + y));
gs.insert(closure_point(0*x + 0*y));
gs.insert(closure_point(0*x + 3*y));
gs.insert(closure_point(3*x + 0*y));
gs.insert(closure_point(3*x + 3*y));
```

Example 5

The following code builds a system of generators having two points and a ray, corresponding to a half-strip in $\mathbb{R}^2$ (the same as Example 2 for the system of constraints):

```
Generator_System gs;
gs.insert(point(0*x + 0*y));
gs.insert(point(0*x + 1*y));
gs.insert(ray(x - y));
```

Note

After inserting a multiset of generators in a generator system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* generator system will be available, where original generators may have been reordered, removed (if they are duplicate or redundant), etc.

Definition at line 183 of file Generator_System.defs.hh.

11.69.2 Constructor & Destructor Documentation

11.69.2.1 Parma_Polyhedra_Library::Generator_System::Generator_System () [inline]

Default constructor: builds an empty system of generators.

Definition at line 32 of file Generator_System.inlines.hh.

Referenced by initialize().

11.69.2.2 Parma_Polyhedra_Library::Generator_System::Generator_System (const Generator & *g*) [inline, explicit]

Builds the singleton system containing only generator `g`.

Definition at line 37 of file Generator_System.inlines.hh.

References insert().

11.69.2.3 Parma_Polyhedra_Library::Generator_System::Generator_System (const Generator_System & *gs*) [inline]

Ordinary copy constructor.

Definition at line 43 of file Generator_System.inlines.hh.

11.69.2.4 Parma_Polyhedra_Library::Generator_System::~Generator_System () [inline]

Destructor.

Definition at line 60 of file Generator_System.inlines.hh.

11.69.2.5 Parma_Polyhedra_Library::Generator_System::Generator_System (Topology *topol*) [inline, explicit, private]

Builds an empty system of generators having the specified topology.

Definition at line 48 of file Generator_System.inlines.hh.

11.69.2.6 Parma_Polyhedra_Library::Generator_System::Generator_System (Topology *topol*, dimension_type *n_rows*, dimension_type *n_columns*) [inline, private]

Builds a system of `n_rows` rays/points on a `n_columns` - 1 dimensional space (including the ϵ dimension, if `topol` is `NOT_NECESSARILY_CLOSED`).

Definition at line 53 of file Generator_System.inlines.hh.

11.69.3 Member Function Documentation

11.69.3.1 void Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points () [private]

For each unmatched point in `*this`, adds the corresponding closure point.

It is assumed that the topology of `*this` is `NOT_NECESSARILY_CLOSED`.

Definition at line 187 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), and OK().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.69.3.2 void Parma_Polyhedra_Library::Generator_System::add_corresponding_points () [private]

For each unmatched closure point in `*this`, adds the corresponding point.

It is assumed that the topology of `*this` is `NOT_NECESSARILY_CLOSED`.

Definition at line 214 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), and OK().

Referenced by Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.69.3.3 bool Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension (Topology *topol*, dimension_type *num_dimensions*) [private]

Adjusts `*this` so that it matches the topology and the number of space dimensions given as parameters (adding or removing columns if needed). Returns `false` if and only if `topol` is equal to `NECESSARILY_CLOSED` and `*this` contains closure points.

Definition at line 40 of file Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::erase_to_end(), has_closure_points(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_System::normalize(), Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), Parma_Polyhedra_Library::Linear_System::set_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), space_dimension(), swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Grid_Generator_System::Grid_Generator_System(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.69.3.4 void Parma_Polyhedra_Library::Generator_System::affine_image (dimension_type *v*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator*) [private]

Assigns to a given variable an affine expression.

Parameters

v Index of the column to which the affine transformation is assigned;

expr The numerator of the affine transformation: $\sum_{i=0}^{n-1} a_i x_i + b$;

denominator The denominator of the affine transformation.

We want to allow affine transformations (see the Introduction) having any rational coefficients. Since the coefficients of the constraints are integers we must also provide an integer `denominator` that will be used as denominator of the affine transformation. The denominator is required to be a positive integer.

The affine transformation assigns to each element of `v` -th column the follow expression:

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{\text{denominator}}.$$

`expr` is a constant parameter and unaltered by this computation.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 788 of file Generator_System.cc.

References Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_-Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_-TEMP_COEFFICIENT, remove_invalid_lines_and_rays(), Parma_Polyhedra_Library::Linear_-Expression::space_dimension(), space_dimension(), Parma_Polyhedra_Library::Linear_System::strong_-normalize(), and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::affine_image(), and Parma_Polyhedra_-Library::Polyhedron::affine_preimage().

11.69.3.5 void Parma_Polyhedra_Library::Generator_System::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 834 of file Generator_System.cc.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_-Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Linear_System::is_-necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_-Library::Generator::LINE, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_-Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_-Library::Generator::RAY, and Parma_Polyhedra_Library::Generator::type().

11.69.3.6 void Parma_Polyhedra_Library::Generator_System::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Referenced by Parma_Polyhedra_Library::Polyhedron::ascii_dump(), and Parma_Polyhedra_-Library::Polyhedron::OK().

11.69.3.7 bool Parma_Polyhedra_Library::Generator_System::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Resizes the matrix of generators using the numbers of rows and columns read from `s`, then initializes the coordinates of each generator and its type reading the contents from `s`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 872 of file Generator_System.cc.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_-Library::Generator::LINE, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_-

Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::POINT, and Parma_Polyhedra_-Library::Generator::RAY.

11.69.3.8 Generator_System::const_iterator Parma_Polyhedra_Library::Generator_-System::begin () const [inline]

Returns the const_iterator pointing to the first generator, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 164 of file Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::begin(), and Parma_Polyhedra_Library::Linear_-System::is_necessarily_closed().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), has_closure_points(), Parma_Polyhedra_-Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), operator<<(), Parma_Polyhedra_Library::Polyhedron::relation_with(), and Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign().

11.69.3.9 void Parma_Polyhedra_Library::Generator_System::clear () [inline]

Removes all the generators from the generator system and sets its space dimension to 0.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 80 of file Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_empty(), and Parma_Polyhedra_-Library::Polyhedron::set_zero_dim_univ().

11.69.3.10 bool Parma_Polyhedra_Library::Generator_System::empty () const [inline]

Returns `true` if and only if `*this` has no generators.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 159 of file Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::has_no_rows().

11.69.3.11 Generator_System::const_iterator Parma_Polyhedra_Library::Generator_-System::end () const [inline]

Returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 172 of file Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::end().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), has_closure_points(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), operator<<(), Parma_Polyhedra_Library::Polyhedron::relation_with(), and Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign().

11.69.3.12 memory_size_type Parma_Polyhedra_Library::Generator_System::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 189 of file Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::external_memory_in_bytes().

11.69.3.13 void Parma_Polyhedra_Library::Generator_System::finalize () [static]

Finalizes the class.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 1017 of file Generator_System.cc.

References zero_dim_univ_p.

11.69.3.14 bool Parma_Polyhedra_Library::Generator_System::has_closure_points () const [private]

Returns `true` if and only if `*this` contains one or more closure points.

Note: the check for the presence of closure points is done under the point of view of the user. Namely, we scan the generator system using high-level iterators, so that closure points that are matching the corresponding points will be disregarded.

Definition at line 235 of file Generator_System.cc.

References begin(), end(), and Parma_Polyhedra_Library::Linear_System::is_necessarily_closed().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), and adjust_topology_and_space_dimension().

11.69.3.15 bool Parma_Polyhedra_Library::Generator_System::has_points () const [private]

Returns `true` if and only if `*this` contains one or more points.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 247 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Polyhedron::OK(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.69.3.16 void Parma_Polyhedra_Library::Generator_System::initialize () [static]

Initializes the class.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 1010 of file Generator_System.cc.

References Generator_System(), Parma_Polyhedra_Library::Generator::zero_dim_point(), and zero_dim_univ_p.

11.69.3.17 void Parma_Polyhedra_Library::Generator_System::insert (const Generator & *g*)

Inserts in `*this` a copy of the generator `g`, increasing the number of space dimensions if needed.

Definition at line 283 of file Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), space_dimension(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Linear_Row::topology(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Generator_System(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), and Parma_Polyhedra_Library::Polyhedron::unconstrain().

11.69.3.18 void Parma_Polyhedra_Library::Generator_System::insert_pending (const Generator & *g*) [private]

Inserts in `*this` a copy of the generator `g`, increasing the number of space dimensions if needed. It is a pending generator.

Definition at line 330 of file Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), space_dimension(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Linear_Row::topology(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), and Parma_Polyhedra_Library::Polyhedron::unconstrain().

11.69.3.19 dimension_type Parma_Polyhedra_Library::Generator_System::max_space_dimension () [inline, static]

Returns the maximum space dimension a Generator_System can handle.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 70 of file Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::max_space_dimension().

11.69.3.20 PPL::dimension_type Parma_Polyhedra_Library::Generator_System::num_lines () const [private]

Returns the number of lines of the system.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 374 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Polyhedron::quick_equivalence_test(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators().

11.69.3.21 PPL::dimension_type Parma_Polyhedra_Library::Generator_System::num_rays () const [private]

Returns the number of rays of the system.

Definition at line 395 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::num_parameters(), and Parma_-Polyhedra_Library::Polyhedron::OK().

11.69.3.22 bool Parma_Polyhedra_Library::Generator_System::OK () const

Checks if all the invariants are satisfied.

Returns `true` if and only if `*this` is a valid Linear_System and each row in the system is a valid Generator.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 1024 of file Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), and Parma_Polyhedra_-Library::Matrix::OK().

Referenced by add_corresponding_closure_points(), add_corresponding_points(), adjust_topology_and_-space_dimension(), insert(), insert_pending(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.69.3.23 Generator_System & Parma_Polyhedra_Library::Generator_System::operator= (const Generator_System & *y*) `[inline]`

Assignment operator.

Definition at line 64 of file Generator_System.inlines.hh.

11.69.3.24 const Generator & Parma_Polyhedra_Library::Generator_System::operator[ ] (dimension_type *k*) const `[inline, private]`

Returns a constant reference to the `k`− th generator of the system.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 90 of file Generator_System.inlines.hh.

References operator[ ]().

11.69.3.25 Generator & Parma_Polyhedra_Library::Generator_System::operator[ ] (dimension_type *k*) `[inline, private]`

Returns the `k`− th generator of the system.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 85 of file Generator_System.inlines.hh.

Referenced by operator[ ]().

11.69.3.26 void Parma_Polyhedra_Library::Generator_System::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

11.69.3.27 PPL::Poly_Con_Relation Parma_Polyhedra_Library::Generator_System::relation_with (const Constraint & *c*) const [private]

Returns the relations holding between the generator system and the constraint `c`.

Definition at line 417 of file Generator_System.cc.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Constraint::space_dimension(), space_dimension(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), Parma_Polyhedra_Library::Generator::type(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Polyhedron::relation_with().

11.69.3.28 void Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays () [private]

Removes all the invalid lines and rays.

The invalid lines and rays are those with all the homogeneous terms set to zero.

Definition at line 950 of file Generator_System.cc.

References Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::swap(), and swap().

Referenced by affine_image(), Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions(), and simplify().

11.69.3.29 bool Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators (const Constraint & *c*) const [private]

Returns `true` if all the generators satisfy `c`.

Definition at line 725 of file Generator_System.cc.

References Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::POINT, space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, Parma_Polyhedra_Library::Generator::type(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::select_CH78_constraints(), and Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign().

11.69.3.30 bool Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators_C (const Constraint & *c*) const [private]

Returns `true` if all the generators satisfy `c`.

It is assumed that `c.is_necessarily_closed()` holds.

11.69.3.31 bool Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators_NNC (const Constraint & *c*) const [private]

Returns `true` if all the generators satisfy `c`.

It is assumed that `c.is_necessarily_closed()` does not hold.

11.69.3.32 void Parma_Polyhedra_Library::Generator_System::simplify () [inline, private]

Applies Gaussian elimination and back-substitution so as to provide a partial simplification of the system of generators.

It is assumed that the system has no pending generators.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 199 of file Generator_System.inlines.hh.

References remove_invalid_lines_and_rays().

11.69.3.33 dimension_type Parma_Polyhedra_Library::Generator_System::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 75 of file Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), adjust_topology_and_space_dimension(), affine_image(), Parma_Polyhedra_Library::Polyhedron::generators(), insert(), insert_pending(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), relation_with(), satisfied_by_all_generators(), Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), and Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible().

11.69.3.34 void Parma_Polyhedra_Library::Generator_System::swap (Generator_System & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 184 of file Generator_System.inlines.hh.

Referenced by adjust_topology_and_space_dimension(), affine_image(), remove_invalid_lines_and_rays(), and swap().

11.69.3.35 memory_size_type Parma_Polyhedra_Library::Generator_System::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 194 of file Generator_System.inlines.hh.

11.69.3.36 const Generator_System & Parma_Polyhedra_Library::Generator_System::zero_dim_univ () [inline, static]

Returns the singleton system containing only Generator::zero_dim_point().

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 178 of file Generator_System.inlines.hh.

References zero_dim_univ_p.

Referenced by Parma_Polyhedra_Library::Polyhedron::generators().

11.69.4 Friends And Related Function Documentation

11.69.4.1 friend class const_iterator [friend]

Definition at line 362 of file Generator_System.defs.hh.

11.69.4.2 std::ostream & operator<< (std::ostream & *s*, const Generator_System & *gs*) [related]

Output operator.

Writes `false` if `gs` is empty. Otherwise, writes on `s` the generators of `gs`, all in one row and separated by ", ".

Definition at line 1043 of file Generator_System.cc.

References begin(), and end().

11.69.4.3 bool operator== (const Polyhedron & *x*, const Polyhedron & *y*) [friend]

11.69.4.4 friend class Parma_Polyhedra_Library::Grid_Generator_System [friend]

Definition at line 364 of file Generator_System.defs.hh.

11.69.4.5 friend class Parma_Polyhedra_Library::Polyhedron [friend]

Definition at line 363 of file Generator_System.defs.hh.

11.69.4.6 void swap (Parma_Polyhedra_Library::Generator_System & *x*, Parma_Polyhedra_Library::Generator_System & *y*) [related]

Specializes `std::swap`.

Definition at line 211 of file Generator_System.inlines.hh.

References swap().

11.69.5 Member Data Documentation

11.69.5.1 const PPL::Generator_System * Parma_Polyhedra_Library::Generator_System::zero_dim_univ_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the singleton system containing only Generator::zero_dim_point().

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 360 of file Generator_System.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_univ().

The documentation for this class was generated from the following files:

- Generator_System.defs.hh
- Generator_System.inlines.hh
- Generator_System.cc

11.70 Parma_Polyhedra_Library::GMP_Integer Class Reference

Unbounded integers as provided by the GMP library.

`#include <GMP_Integer.types.hh>`

Related Functions

(Note that these are not member functions.)

Accessor Functions

- const mpz_class & raw_value (const GMP_Integer &x)
  Returns a const reference to the underlying integer value.
- mpz_class & raw_value (GMP_Integer &x)
  Returns a reference to the underlying integer value.

Arithmetic Operators

- void neg_assign (GMP_Integer &x)
  Assigns to `x` *its negation.*
- void neg_assign (GMP_Integer &x, const GMP_Integer &y)
  Assigns to `x` *the negation of* `y`.
- void abs_assign (GMP_Integer &x)
  Assigns to `x` *its absolute value.*
- void abs_assign (GMP_Integer &x, const GMP_Integer &y)
  Assigns to `x` *the absolute value of* `y`.
- void rem_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  Assigns to `x` *the remainder of the division of* `y` *by* `z`.
- void gcd_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  Assigns to `x` *the greatest common divisor of* `y` *and* `z`.
- void gcdext_assign (GMP_Integer &x, GMP_Integer &s, GMP_Integer &t, const GMP_Integer &y, const GMP_Integer &z)

Extended GCD.

- void lcm_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  Assigns to `x` *the least common multiple of* `y` *and* `z`.
- void add_mul_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  Assigns to `x` *the value* `x + y * z`.
- void sub_mul_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  Assigns to `x` *the value* `x - y * z`.
- void mul_2exp_assign (GMP_Integer &x, const GMP_Integer &y, unsigned int exp)
  Assigns to `x` *the value* $y \cdot 2^{\texttt{exp}}$.
- void div_2exp_assign (GMP_Integer &x, const GMP_Integer &y, unsigned int exp)
  Assigns to `x` *the value* $y/2^{\texttt{exp}}$.
- void exact_div_assign (GMP_Integer &x, const GMP_Integer &y, const GMP_Integer &z)
  If `z` *divides* `y`*, assigns to* `x` *the quotient of the integer division of* `y` *and* `z`.
- void sqrt_assign (GMP_Integer &x, const GMP_Integer &y)
  Assigns to `x` *the integer square root of* `y`.
- int cmp (const GMP_Integer &x, const GMP_Integer &y)
  Returns a negative, zero or positive value depending on whether `x` *is lower than, equal to or greater than* `y`*, respectively.*

11.70.1 Detailed Description

Unbounded integers as provided by the GMP library. GMP_Integer is an alias for the `mpz_class` type defined in the C++ interface of the GMP library. For more information, see http://www.swox.com/gmp/

11.70.2 Friends And Related Function Documentation

11.70.2.1 void abs_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Assigns to `x` the absolute value of `y`.

Definition at line 47 of file GMP_Integer.inlines.hh.

11.70.2.2 void abs_assign (GMP_Integer & *x*) [related]

Assigns to `x` its absolute value.

Definition at line 42 of file GMP_Integer.inlines.hh.

11.70.2.3 void add_mul_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Assigns to x the value x + y * z.

Definition at line 75 of file GMP_Integer.inlines.hh.

11.70.2.4 int cmp (const GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Returns a negative, zero or positive value depending on whether x is lower than, equal to or greater than y, respectively.

Definition at line 106 of file GMP_Integer.inlines.hh.

11.70.2.5 void div_2exp_assign (GMP_Integer & *x*, const GMP_Integer & *y*, unsigned int *exp*) [related]

Assigns to x the value $y/2^{\mathtt{exp}}$.

Definition at line 90 of file GMP_Integer.inlines.hh.

11.70.2.6 void exact_div_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

If z divides y, assigns to x the quotient of the integer division of y and z.

The behavior is undefined if z does not divide y.

Definition at line 95 of file GMP_Integer.inlines.hh.

11.70.2.7 void gcd_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Assigns to x the greatest common divisor of y and z.

Definition at line 52 of file GMP_Integer.inlines.hh.

11.70.2.8 void gcdext_assign (GMP_Integer & *x*, GMP_Integer & *s*, GMP_Integer & *t*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Extended GCD.

Assigns to x the greatest common divisor of y and z, and to s and t the values such that y * s + z * t = x.

Definition at line 62 of file GMP_Integer.inlines.hh.

11.70.2.9 void lcm_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Assigns to x the least common multiple of y and z.

Definition at line 70 of file GMP_Integer.inlines.hh.

11.70.2.10 void mul_2exp_assign (GMP_Integer & *x*, const GMP_Integer & *y*, unsigned int *exp*) [related]

Assigns to x the value $y \cdot 2^{\texttt{exp}}$.

Definition at line 85 of file GMP_Integer.inlines.hh.

11.70.2.11 void neg_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Assigns to x the negation of y.

Definition at line 37 of file GMP_Integer.inlines.hh.

11.70.2.12 void neg_assign (GMP_Integer & *x*) [related]

Assigns to x its negation.

Definition at line 32 of file GMP_Integer.inlines.hh.

11.70.2.13 mpz_class & raw_value (GMP_Integer & *x*) [related]

Returns a reference to the underlying integer value.

Definition at line 116 of file GMP_Integer.inlines.hh.

11.70.2.14 const mpz_class & raw_value (const GMP_Integer & *x*) [related]

Returns a const reference to the underlying integer value.

Definition at line 111 of file GMP_Integer.inlines.hh.

11.70.2.15 void rem_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Assigns to x the remainder of the division of y by z.

Definition at line 57 of file GMP_Integer.inlines.hh.

11.70.2.16 void sqrt_assign (GMP_Integer & *x*, const GMP_Integer & *y*) [related]

Assigns to `x` the integer square root of `y`.

Definition at line 101 of file GMP_Integer.inlines.hh.

11.70.2.17 void sub_mul_assign (GMP_Integer & *x*, const GMP_Integer & *y*, const GMP_Integer & *z*) [related]

Assigns to `x` the value `x - y * z`.

Definition at line 80 of file GMP_Integer.inlines.hh.

The documentation for this class was generated from the following files:

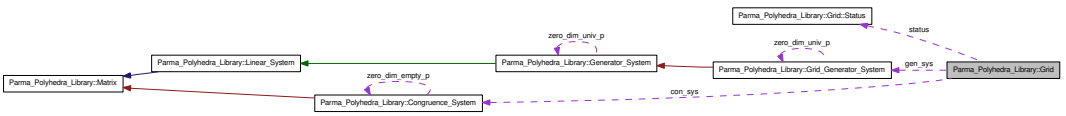
- GMP_Integer.types.hh
- GMP_Integer.defs.hh

11.71 Parma_Polyhedra_Library::Grid Class Reference

A grid.

`#include <Grid.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Grid:



Classes

- class Status

  A conjunctive assertion about a grid.

Public Types

- typedef Coefficient coefficient_type

  The numeric type of coefficients.

Public Member Functions

- Grid (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

Builds a grid having the specified properties.

- Grid (const Congruence_System &cgs)

  Builds a grid, copying a system of congruences.

- Grid (Congruence_System &cgs, Recycle_Input dummy)

  Builds a grid, recycling a system of congruences.

- Grid (const Constraint_System &cs)

  Builds a grid, copying a system of constraints.

- Grid (Constraint_System &cs, Recycle_Input dummy)

  Builds a grid, recycling a system of constraints.

- Grid (const Grid_Generator_System &const_gs)

  Builds a grid, copying a system of grid generators.

- Grid (Grid_Generator_System &gs, Recycle_Input dummy)

  Builds a grid, recycling a system of grid generators.

- template<typename Interval >
  Grid (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a grid out of a box.

- template<typename U >
  Grid (const BD_Shape< U > &bd, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a grid out of a bounded-difference shape.

- template<typename U >
  Grid (const Octagonal_Shape< U > &os, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a grid out of an octagonal shape.

- Grid (const Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a grid from a polyhedron using algorithms whose complexity does not exceed the one specified by `complexity`*. If* `complexity` *is* `ANY_COMPLEXITY`*, then the grid built is the smallest one containing* `ph`*.*

- Grid (const Grid &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- Grid & operator= (const Grid &y)

  The assignment operator. (`*this` *and* `y` *can be dimension-incompatible.).*

Member Functions that Do Not Modify the Grid

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`*.*

- dimension_type affine_dimension () const

  Returns 0*, if* `*this` *is empty; otherwise, returns the affine dimension of* `*this`*.*

- Constraint_System constraints () const
  Returns a system of equality constraints satisfied by `*this` *with the same affine dimension as* `*this`.
- Constraint_System minimized_constraints () const
  Returns a minimal system of equality constraints satisfied by `*this` *with the same affine dimension as* `*this`.
- const Congruence_System & congruences () const
  Returns the system of congruences.
- const Congruence_System & minimized_congruences () const
  Returns the system of congruences in minimal form.
- const Grid_Generator_System & grid_generators () const
  Returns the system of generators.
- const Grid_Generator_System & minimized_grid_generators () const
  Returns the minimized system of generators.
- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between `*this` *and* `cg`.
- Poly_Gen_Relation relation_with (const Grid_Generator &g) const
  Returns the relations holding between `*this` *and* `g`.
- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between `*this` *and* `g`.
- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between `*this` *and* `c`.
- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty grid.*
- bool is_universe () const
  Returns `true` *if and only if* `*this` *is a universe grid.*
- bool is_topologically_closed () const
  Returns `true` *if and only if* `*this` *is a topologically closed subset of the vector space.*
- bool is_disjoint_from (const Grid &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*
- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*
- bool is_bounded () const
  Returns `true` *if and only if* `*this` *is bounded.*
- bool contains_integer_point () const
  Returns `true` *if and only if* `*this` *contains at least one integer point.*
- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.

- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded in* `*this`.

- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded in* `*this`.

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &point) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &point) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool frequency (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const
  Returns `true` *if and only if* `*this` *is not empty and frequency for* `*this` *with respect to* `expr` *is defined, in which case the frequency and the value for* `expr` *that is closest to zero are computed.*

- bool contains (const Grid &y) const
  Returns `true` *if and only if* `*this` *contains* `y`.

- bool strictly_contains (const Grid &y) const
  Returns `true` *if and only if* `*this` *strictly contains* `y`.

- bool OK (bool check_not_empty=false) const
  Checks if all the invariants are satisfied.

Space Dimension Preserving Member Functions that May Modify the Grid

- void add_congruence (const Congruence &cg)
  Adds a copy of congruence `cg` *to* `*this`.

- void add_grid_generator (const Grid_Generator &g)
  Adds a copy of grid generator `g` *to the system of generators of* `*this`.

- void add_congruences (const Congruence_System &cgs)
  Adds a copy of each congruence in `cgs` *to* `*this`.

- void add_recycled_congruences (Congruence_System &cgs)
  Adds the congruences in `cgs` *to* *this.

- void add_constraint (const Constraint &c)
  Adds to `*this` *a congruence equivalent to constraint* `c`.
- void add_constraints (const Constraint_System &cs)
  Adds to `*this` *congruences equivalent to the constraints in* `cs`.
- void add_recycled_constraints (Constraint_System &cs)
  Adds to `*this` *congruences equivalent to the constraints in* `cs`.
- void refine_with_congruence (const Congruence &cg)
  Uses a copy of the congruence `cg` *to refine* `*this`.
- void refine_with_congruences (const Congruence_System &cgs)
  Uses a copy of the congruences in `cgs` *to refine* `*this`.
- void refine_with_constraint (const Constraint &c)
  Uses a copy of the constraint `c` *to refine* `*this`.
- void refine_with_constraints (const Constraint_System &cs)
  Uses a copy of the constraints in `cs` *to refine* `*this`.
- void add_grid_generators (const Grid_Generator_System &gs)
  Adds a copy of the generators in `gs` *to the system of generators of* `*this`.
- void add_recycled_grid_generators (Grid_Generator_System &gs)
  Adds the generators in `gs` *to the system of generators of* `this`.
- void unconstrain (Variable var)
  Computes the cylindrification of `*this` *with respect to space dimension* `var`*, assigning the result to* `*this`.
- void unconstrain (const Variables_Set &vars)
  Computes the cylindrification of `*this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `*this`.
- void intersection_assign (const Grid &y)
  Assigns to `*this` *the intersection of* `*this` *and* `y`.
- void upper_bound_assign (const Grid &y)
  Assigns to `*this` *the least upper bound of* `*this` *and* `y`.
- bool upper_bound_assign_if_exact (const Grid &y)
  If the upper bound of `*this` *and* `y` *is exact it is assigned to* `this` *and* `true` *is returned, otherwise* `false` *is returned.*
- void difference_assign (const Grid &y)
  Assigns to `*this` *the grid-difference of* `*this` *and* `y`.
- bool simplify_using_context_assign (const Grid &y)
  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`*. If* `false` *is returned, then the intersection is empty.*
- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())
  Assigns to `*this` *the affine image of* `this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())

  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one(), Coefficient_-traits::const_reference modulus=Coefficient_zero())

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' = \frac{\mathrm{expr}}{\mathrm{denominator}} \pmod{\mathrm{modulus}}$.

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_-Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one(), Coefficient_traits::const_reference modulus=Coefficient_zero())

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' = \frac{\mathrm{expr}}{\mathrm{denominator}} \pmod{\mathrm{modulus}}$.

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs, Coefficient_traits::const_reference modulus=Coefficient_zero())

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' = \mathrm{rhs} \pmod{\mathrm{modulus}}$.

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs, Coefficient_traits::const_reference modulus=Coefficient_-zero())

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' = \mathrm{rhs} \pmod{\mathrm{modulus}}$.

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_-Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_-Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void time_elapse_assign (const Grid &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`.

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_-Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)

  Wraps the specified dimensions of the vector space.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping all points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping all points with non-integer coordinates for the space dimensions corresponding to* `vars`.

- void topological_closure_assign ()
  Assigns to `*this` *its topological closure.*

- void congruence_widening_assign (const Grid &y, unsigned *tp=NULL)
  Assigns to `*this` *the result of computing the Grid widening between* `*this` *and* `y` *using congruence systems.*

- void generator_widening_assign (const Grid &y, unsigned *tp=NULL)
  Assigns to `*this` *the result of computing the Grid widening between* `*this` *and* `y` *using generator systems.*

- void widening_assign (const Grid &y, unsigned *tp=NULL)
  Assigns to `*this` *the result of computing the Grid widening between* `*this` *and* `y`.

- void limited_congruence_extrapolation_assign (const Grid &y, const Congruence_System &cgs, unsigned *tp=NULL)
  Improves the result of the congruence variant of Grid widening computation by also enforcing those congruences in `cgs` *that are satisfied by all the points of* `*this`.

- void limited_generator_extrapolation_assign (const Grid &y, const Congruence_System &cgs, unsigned *tp=NULL)
  Improves the result of the generator variant of the Grid widening computation by also enforcing those congruences in `cgs` *that are satisfied by all the points of* `*this`.

- void limited_extrapolation_assign (const Grid &y, const Congruence_System &cgs, unsigned *tp=NULL)
  Improves the result of the Grid widening computation by also enforcing those congruences in `cgs` *that are satisfied by all the points of* `*this`.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)
  Adds `m` *new space dimensions and embeds the old grid in the new vector space.*

- void add_space_dimensions_and_project (dimension_type m)
  Adds `m` *new space dimensions to the grid and does not embed it in the new vector space.*

- void concatenate_assign (const Grid &y)
  Assigns to `*this` *the concatenation of* `*this` *and* `y`*, taken in this order.*

- void remove_space_dimensions (const Variables_Set &vars)
  Removes all the specified dimensions from the vector space.

- void remove_higher_space_dimensions (dimension_type new_dimension)
  Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`*..*

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)
  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)
  Creates `m` *copies of the space dimension corresponding to* `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` into `dest`.

Miscellaneous Member Functions

- ~Grid ()
  Destructor.
- void swap (Grid &y)
  Swaps `*this` with grid `y`. (`*this` and `y` can be dimension-incompatible.).
- void ascii_dump () const
  Writes to `std::cerr` an ASCII representation of `*this`.
- void ascii_dump (std::ostream &s) const
  Writes to `s` an ASCII representation of `*this`.
- void print () const
  Prints `*this` to `std::cerr` using `operator<<`.
- bool ascii_load (std::istream &s)
  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.
- memory_size_type total_memory_in_bytes () const
  Returns the total size in bytes of the memory occupied by `*this`.
- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.
- int32_t hash_code () const
  Returns a 32-bit hash code for `*this`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension all kinds of Grid can handle.
- static bool can_recycle_congruence_systems ()
  Returns true indicating that this domain has methods that can recycle congruences.
- static bool can_recycle_constraint_systems ()
  Returns true indicating that this domain has methods that can recycle constraints.

Private Types

- enum Dimension_Kind {
  PARAMETER, LINE, GEN_VIRTUAL, PROPER_CONGRUENCE = PARAMETER,
  CON_VIRTUAL = LINE, EQUALITY = GEN_VIRTUAL }
- enum Three_Valued_Boolean { TVB_TRUE, TVB_FALSE, TVB_DONT_KNOW }
- typedef std::vector< Dimension_Kind > Dimension_Kinds

Private Member Functions

- void construct (dimension_type num_dimensions, Degenerate_Element kind)

  Builds a grid universe or empty grid.

- void construct (Congruence_System &cgs)

  Builds a grid from a system of congruences.

- void construct (Grid_Generator_System &ggs)

  Builds a grid from a system of grid generators.

- Three_Valued_Boolean quick_equivalence_test (const Grid &y) const

  Polynomial but incomplete equivalence test between grids.

- bool is_included_in (const Grid &y) const

  Returns `true` *if and only if* `*this` *is included in* `y`.

- bool bounds (const Linear_Expression &expr, const char *method_call) const

  Checks if and how `expr` *is bounded in* `*this`.

- bool max_min (const Linear_Expression &expr, const char *method_call, Coefficient &ext_n, Coefficient &ext_d, bool &included, Generator *point=NULL) const

  Maximizes or minimizes `expr` *subject to* `*this`.

- bool frequency_no_check (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const

  Returns `true` *if and only if* `*this` *is not empty and frequency for* `*this` *with respect to* `expr` *is defined, in which case the frequency and the value for* `expr` *that is closest to zero are computed.*

- bool bounds_no_check (const Linear_Expression &expr) const

  Checks if and how `expr` *is bounded in* `*this`.

- void add_congruence_no_check (const Congruence &cg)

  Adds the congruence `cg` *to* `*this`.

- void add_constraint_no_check (const Constraint &c)

  Uses the constraint `c` *to refine* `*this`.

- void refine_no_check (const Constraint &c)

  Uses the constraint `c` *to refine* `*this`.

- void add_space_dimensions (Congruence_System &cgs, Grid_Generator_System &gs, dimension_type dims)

  Adds new space dimensions to the given systems.

- void add_space_dimensions (Grid_Generator_System &gs, Congruence_System &cgs, dimension_type dims)

  Adds new space dimensions to the given systems.

Private Verifiers: Verify if Individual Flags are Set

- bool marked_empty () const
  Returns `true` *if the grid is known to be empty.*

- bool congruences_are_up_to_date () const
  Returns `true` *if the system of congruences is up-to-date.*

- bool generators_are_up_to_date () const
  Returns `true` *if the system of generators is up-to-date.*

- bool congruences_are_minimized () const
  Returns `true` *if the system of congruences is minimized.*

- bool generators_are_minimized () const
  Returns `true` *if the system of generators is minimized.*

State Flag Setters: Set Only the Specified Flags

- void set_zero_dim_univ ()
  Sets `status` *to express that the grid is the universe 0-dimension vector space, clearing all corresponding matrices.*

- void set_empty ()
  Sets `status` *to express that the grid is empty, clearing all corresponding matrices.*

- void set_congruences_up_to_date ()
  Sets `status` *to express that congruences are up-to-date.*

- void set_generators_up_to_date ()
  Sets `status` *to express that generators are up-to-date.*

- void set_congruences_minimized ()
  Sets `status` *to express that congruences are minimized.*

- void set_generators_minimized ()
  Sets `status` *to express that generators are minimized.*

State Flag Cleaners: Clear Only the Specified Flag

- void clear_empty ()
  Clears the `status` *flag indicating that the grid is empty.*

- void clear_congruences_up_to_date ()
  Sets `status` *to express that congruences are out of date.*

- void clear_generators_up_to_date ()
  Sets `status` *to express that generators are out of date.*

- void clear_congruences_minimized ()
  Sets `status` *to express that congruences are no longer minimized.*

- void clear_generators_minimized ()
  Sets `status` *to express that generators are no longer minimized.*

Updating Matrices

- void update_congruences () const
  Updates and minimizes the congruences from the generators.
- bool update_generators () const
  Updates and minimizes the generators from the congruences.

Minimization of Descriptions

- bool minimize () const
  Minimizes both the congruences and the generators.

Widening- and Extrapolation-Related Functions

- void select_wider_congruences (const Grid &y, Congruence_System &selected_cgs) const
  Copies a widened selection of congruences from `y` *to* `selected_cgs`.
- void select_wider_generators (const Grid &y, Grid_Generator_System &widened_ggs) const
  Copies widened generators from `y` *to* `widened_ggs`.

Static Private Member Functions

Minimization-related Static Member Functions

- static void normalize_divisors (Grid_Generator_System &sys, Coefficient &divisor, const Grid_Generator *first_point=NULL)
  Normalizes the divisors in `sys`.
- static void normalize_divisors (Grid_Generator_System &sys)
  Normalizes the divisors in `sys`.
- static void normalize_divisors (Grid_Generator_System &sys, Grid_Generator_System &gen_sys)
  Normalize all the divisors in `sys` *and* `gen_sys`.
- static void conversion (Congruence_System &source, Grid_Generator_System &dest, Dimension_Kinds &dim_kinds)
  Converts generator system `dest` *to be equivalent to congruence system* `source`.
- static void conversion (Grid_Generator_System &source, Congruence_System &dest, Dimension_Kinds &dim_kinds)
  Converts congruence system `dest` *to be equivalent to generator system* `source`.
- static bool simplify (Congruence_System &cgs, Dimension_Kinds &dim_kinds)
  Converts `cgs` *to upper triangular (i.e. minimized) form.*
- static void simplify (Grid_Generator_System &gs, Dimension_Kinds &dim_kinds)
  Converts `gs` *to lower triangular (i.e. minimized) form.*
- static void reduce_line_with_line (Grid_Generator &row, Grid_Generator &pivot, dimension_type col)

Reduces the line `row` using the line `pivot`.

- static void reduce_equality_with_equality (Congruence &row, const Congruence &pivot, dimension_type col)
  Reduces the equality `row` using the equality `pivot`.

- template<typename R >
  static void reduce_pc_with_pc (R &row, R &pivot, dimension_type col, dimension_type start, dimension_type end)
  Reduces `row` using `pivot`.

- static void reduce_parameter_with_line (Grid_Generator &row, const Grid_Generator &pivot, dimension_type col, Grid_Generator_System &sys)
  Reduce `row` using `pivot`.

- static void reduce_congruence_with_equality (Congruence &row, const Congruence &pivot, dimension_type col, Congruence_System &sys)
  Reduce `row` using `pivot`.

- template<typename M , typename R >
  static void reduce_reduced (M &sys, dimension_type dim, dimension_type pivot_index, dimension_type start, dimension_type end, const Dimension_Kinds &dim_kinds, bool generators=true)
  Reduce column `dim` in rows preceding `pivot_index` in `sys`.

- static void multiply_grid (const Coefficient &multiplier, Congruence &cg, Congruence_System &dest, dimension_type num_rows, dimension_type num_dims)
  Multiply the elements of `dest` by `multiplier`.

- static void multiply_grid (const Coefficient &multiplier, Grid_Generator &gen, Grid_Generator_System &dest, dimension_type num_rows, dimension_type num_dims)
  Multiply the elements of `dest` by `multiplier`.

- static bool lower_triangular (const Congruence_System &sys, const Dimension_Kinds &dim_kinds)
  If `sys` is lower triangular return `true`, else return `false`.

- static bool upper_triangular (const Grid_Generator_System &sys, const Dimension_Kinds &dim_kinds)
  If `sys` is upper triangular return `true`, else return `false`.

- template<typename M , typename R >
  static bool rows_are_zero (M &system, dimension_type first, dimension_type last, dimension_type row_size)
  Checks that trailing rows contain only zero terms.

Private Attributes

- Congruence_System con_sys
  The system of congruences.

- Grid_Generator_System gen_sys
  The system of generators.

- Status status

  The status flags to keep track of the grid's internal state.

- dimension_type space_dim

  The number of dimensions of the enclosing vector space.

- Dimension_Kinds dim_kinds

Friends

- class Parma_Polyhedra_Library::Grid_Certificate
- class Parma_Polyhedra_Library::Box
- bool operator== (const Grid &x, const Grid &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same grid.*

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Grid &gr)

  Output operator.

- bool operator!= (const Grid &x, const Grid &y)

  Returns `true` *if and only if* `x` *and* `y` *are different grids.*

- void swap (Parma_Polyhedra_Library::Grid &x, Parma_Polyhedra_Library::Grid &y)

  Specializes `std::swap`.

Exception Throwers

- void throw_runtime_error (const char *method) const
- void throw_invalid_argument (const char *method, const char *reason) const
- void throw_dimension_incompatible (const char *method, const char *other_name, dimension_type other_dim) const
- void throw_dimension_incompatible (const char *method, const char *gr_name, const Grid &gr) const
- void throw_dimension_incompatible (const char *method, const char *e_name, const Linear_Expression &e) const
- void throw_dimension_incompatible (const char *method, const char *cg_name, const Congruence &cg) const
- void throw_dimension_incompatible (const char *method, const char *c_name, const Constraint &c) const
- void throw_dimension_incompatible (const char *method, const char *g_name, const Grid_Generator &g) const
- void throw_dimension_incompatible (const char *method, const char *g_name, const Generator &g) const

- void throw_dimension_incompatible (const char *method, const char *cgs_name, const Congruence_System &cgs) const
- void throw_dimension_incompatible (const char *method, const char *cs_name, const Constraint_System &cs) const
- void throw_dimension_incompatible (const char *method, const char *gs_name, const Grid_Generator_System &gs) const
- void throw_dimension_incompatible (const char *method, const char *var_name, Variable var) const
- void throw_dimension_incompatible (const char *method, dimension_type required_space_dim) const
- void throw_invalid_constraint (const char *method, const char *c_name) const
- void throw_invalid_constraints (const char *method, const char *cs_name) const
- void throw_invalid_generator (const char *method, const char *g_name) const
- void throw_invalid_generators (const char *method, const char *gs_name) const
- static void throw_space_dimension_overflow (const char *method, const char *reason)

11.71.1 Detailed Description

A grid. An object of the class Grid represents a rational grid.

The domain of grids *optimally supports*:

- all (proper and non-proper) congruences;
- tautological and inconsistent constraints;
- linear equality constraints (i.e., non-proper congruences).

Depending on the method, using a constraint that is not optimally supported by the domain will either raise an exception or result in a (possibly non-optimal) upward approximation.

The domain of grids support a concept of double description similar to the one developed for polyhedra: hence, a grid can be specified as either a finite system of congruences or a finite system of generators (see Section Rational Grids) and it is always possible to obtain either representation. That is, if we know the system of congruences, we can obtain from this a system of generators that define the same grid and vice versa. These systems can contain redundant members, or they can be in the minimal form.

A key attribute of any grid is its space dimension (the dimension $n \in \mathbb{N}$ of the enclosing vector space):

- all grids, the empty ones included, are endowed with a space dimension;
- most operations working on a grid and another object (another grid, a congruence, a generator, a set of variables, etc.) will throw an exception if the grid and the object are not dimension-compatible (see Section Space Dimensions and Dimension-compatibility for Grids);
- the only ways in which the space dimension of a grid can be changed are with *explicit* calls to operators provided for that purpose, and with standard copy, assignment and swap operators.

Note that two different grids can be defined on the zero-dimension space: the empty grid and the universe grid R^0.

In all the examples it is assumed that variables `x` and `y` are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a grid corresponding to the even integer pairs in $\mathbb{R}^2$, given as a system of congruences:

```
Congruence_System cgs;
cgs.insert((x %= 0) / 2);
cgs.insert((y %= 0) / 2);
Grid gr(cgs);
```

The following code builds the same grid as above, but starting from a system of generators specifying three of the points:

```
Grid_Generator_System gs;
gs.insert(grid_point(0*x + 0*y));
gs.insert(grid_point(0*x + 2*y));
gs.insert(grid_point(2*x + 0*y));
Grid gr(gs);
```

Example 2

The following code builds a grid corresponding to a line in $\mathbb{R}^2$ by adding a single congruence to the universe grid:

```
Congruence_System cgs;
cgs.insert(x - y == 0);
Grid gr(cgs);
```

The following code builds the same grid as above, but starting from a system of generators specifying a point and a line:

```
Grid_Generator_System gs;
gs.insert(grid_point(0*x + 0*y));
gs.insert(grid_line(x + y));
Grid gr(gs);
```

Example 3

The following code builds a grid corresponding to the integral points on the line $x = y$ in $\mathbb{R}^2$ constructed by adding an equality and congruence to the universe grid:

```
Congruence_System cgs;
cgs.insert(x - y == 0);
cgs.insert(x %= 0);
Grid gr(cgs);
```

The following code builds the same grid as above, but starting from a system of generators specifying a point and a parameter:

```
Grid_Generator_System gs;
gs.insert(grid_point(0*x + 0*y));
gs.insert(parameter(x + y));
Grid gr(gs);
```

Example 4

The following code builds the grid corresponding to a plane by creating the universe grid in $\mathbb{R}^2$:

```
Grid gr(2);
```

The following code builds the same grid as above, but starting from the empty grid in $\mathbb{R}^2$ and inserting the appropriate generators (a point, and two lines).

```
Grid gr(2, EMPTY);
gr.add_grid_generator(grid_point(0*x + 0*y));
gr.add_grid_generator(grid_line(x));
gr.add_grid_generator(grid_line(y));
```

Note that a generator system must contain a point when describing a grid. To ensure that this is always the case it is required that the first generator inserted in an empty grid is a point (otherwise, an exception is thrown).

Example 5

The following code shows the use of the function `add_space_dimensions_and_embed`:

```
Grid gr(1);
gr.add_congruence(x == 2);
gr.add_space_dimensions_and_embed(1);
```

We build the universe grid in the 1-dimension space $\mathbb{R}$. Then we add a single equality congruence, thus obtaining the grid corresponding to the singleton set $\{2\} \subseteq \mathbb{R}$. After the last line of code, the resulting grid is

$$\left\{ (2, y)^{\mathrm{T}} \in \mathbb{R}^2 \;\middle|\; y \in \mathbb{R} \right\}.$$

Example 6

The following code shows the use of the function `add_space_dimensions_and_project`:

```
Grid gr(1);
gr.add_congruence(x == 2);
gr.add_space_dimensions_and_project(1);
```

The first two lines of code are the same as in Example 4 for `add_space_dimensions_and_embed`. After the last line of code, the resulting grid is the singleton set $\left\{(2, 0)^{\mathrm{T}}\right\} \subseteq \mathbb{R}^2$.

Example 7

The following code shows the use of the function `affine_image`:

```
Grid gr(2, EMPTY);
gr.add_grid_generator(grid_point(0*x + 0*y));
gr.add_grid_generator(grid_point(4*x + 0*y));
gr.add_grid_generator(grid_point(0*x + 2*y));
Linear_Expression expr = x + 3;
gr.affine_image(x, expr);
```

In this example the starting grid is all the pairs of x and y in $\mathbb{R}^2$ where x is an integer multiple of 4 and y is an integer multiple of 2. The considered variable is x and the affine expression is $x + 3$. The resulting grid is the given grid translated 3 integers to the right (all the pairs (x, y) where x is -1 plus an integer multiple of 4 and y is an integer multiple of 2). Moreover, if the affine transformation for the same variable `x` is instead $x + y$:

```
Linear_Expression expr = x + y;
```

the resulting grid is every second integral point along the $x = y$ line, with this line of points repeated at every fourth integral value along the x axis. Instead, if we do not use an invertible transformation for the same variable; for example, the affine expression y:

```
Linear_Expression expr = y;
```

the resulting grid is every second point along the $x = y$ line.

Example 8

The following code shows the use of the function `affine_preimage`:

```
Grid gr(2, EMPTY);
gr.add_grid_generator(grid_point(0*x + 0*y));
gr.add_grid_generator(grid_point(4*x + 0*y));
gr.add_grid_generator(grid_point(0*x + 2*y));
Linear_Expression expr = x + 3;
gr.affine_preimage(x, expr);
```

In this example the starting grid, `var` and the affine expression and the denominator are the same as in Example 6, while the resulting grid is similar but translated 3 integers to the left (all the pairs (x, y) where x is -3 plus an integer multiple of 4 and y is an integer multiple of 2).. Moreover, if the affine transformation for `x` is $x + y$

```
Linear_Expression expr = x + y;
```

the resulting grid is a similar grid to the result in Example 6, only the grid is slanted along $x = -y$. Instead, if we do not use an invertible transformation for the same variable `x`, for example, the affine expression y:

```
Linear_Expression expr = y;
```

the resulting grid is every fourth line parallel to the x axis.

Example 9

For this example we also use the variables:

```
Variable z(2);
Variable w(3);
```

The following code shows the use of the function `remove_space_dimensions`:

```
Grid_Generator_System gs;
gs.insert(grid_point(3*x + y +0*z + 2*w));
Grid gr(gs);
Variables_Set vars;
vars.insert(y);
vars.insert(z);
gr.remove_space_dimensions(vars);
```

The starting grid is the singleton set $\{(3, 1, 0, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^4$, while the resulting grid is $\{(3, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$. Be careful when removing space dimensions *incrementally*: since dimensions are automatically renamed after each application of the `remove_space_dimensions` operator, unexpected results can be obtained. For instance, by using the following code we would obtain a different result:

```
set<Variable> vars1;
vars1.insert(y);
gr.remove_space_dimensions(vars1);
set<Variable> vars2;
vars2.insert(z);
gr.remove_space_dimensions(vars2);
```

In this case, the result is the grid $\{(3, 0)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$: when removing the set of dimensions `vars2` we are actually removing variable w of the original grid. For the same reason, the operator `remove_space_dimensions` is not idempotent: removing twice the same non-empty set of dimensions is never the same as removing them just once.

Definition at line 359 of file Grid.defs.hh.

11.71.2 Member Typedef Documentation

11.71.2.1 typedef Coefficient Parma_Polyhedra_Library::Grid::coefficient_type

The numeric type of coefficients.

Definition at line 362 of file Grid.defs.hh.

11.71.2.2 typedef std::vector<Dimension_Kind> Parma_Polyhedra_Library::Grid::Dimension_Kinds [private]

Definition at line 1990 of file Grid.defs.hh.

11.71.3 Member Enumeration Documentation

11.71.3.1 enum Parma_Polyhedra_Library::Grid::Dimension_Kind [private]

Enumerator:

PARAMETER
LINE
GEN_VIRTUAL
PROPER_CONGRUENCE
CON_VIRTUAL
EQUALITY

Definition at line 1981 of file Grid.defs.hh.

11.71.3.2 enum Parma_Polyhedra_Library::Grid::Three_Valued_Boolean [private]

Enumerator:

TVB_TRUE
TVB_FALSE
TVB_DONT_KNOW

Definition at line 2138 of file Grid.defs.hh.

11.71.4 Constructor & Destructor Documentation

11.71.4.1 Parma_Polyhedra_Library::Grid::Grid (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds a grid having the specified properties.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the grid;

kind Specifies whether the universe or the empty grid has to be built.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 122 of file Grid.inlines.hh.

References construct(), and OK().

11.71.4.2 Parma_Polyhedra_Library::Grid::Grid (const Congruence_System & *cgs*) [inline, explicit]

Builds a grid, copying a system of congruences.

The grid inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the grid.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 136 of file Grid.inlines.hh.

References construct().

11.71.4.3 Parma_Polyhedra_Library::Grid::Grid (Congruence_System & *cgs*, Recycle_Input *dummy*) [inline]

Builds a grid, recycling a system of congruences.

The grid inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the grid. Its data-structures may be recycled to build the grid.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 149 of file Grid.inlines.hh.

References construct().

11.71.4.4 Parma_Polyhedra_Library::Grid::Grid (const Constraint_System & *cs*) [explicit]

Builds a grid, copying a system of constraints.

The grid inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the grid.

Exceptions

std::invalid_argument Thrown if the constraint system `cs` contains inequality constraints.

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 60 of file Grid_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), con_sys, construct(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Congruence_System::insert(), OK(), Parma_Polyhedra_Library::Grid::Status::set_empty(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, throw_invalid_constraints(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

11.71.4.5 Parma_Polyhedra_Library::Grid::Grid (Constraint_System & *cs*, Recycle_Input *dummy*)

Builds a grid, recycling a system of constraints.

The grid inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the grid. Its data-structures may be recycled to build the grid.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the constraint system `cs` contains inequality constraints.

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 99 of file Grid_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), con_sys, construct(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Congruence_System::insert(), OK(), Parma_Polyhedra_Library::Grid::Status::set_empty(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, throw_invalid_constraint(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

11.71.4.6 Parma_Polyhedra_Library::Grid::Grid (const Grid_Generator_System & *const_gs*) [inline, explicit]

Builds a grid, copying a system of grid generators.

The grid inherits the space dimension of the generator system.

Parameters

const_gs The system of generators defining the grid.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 161 of file Grid.inlines.hh.

References construct().

11.71.4.7 Parma_Polyhedra_Library::Grid::Grid (Grid_Generator_System & *gs*, Recycle_Input *dummy*) [inline]

Builds a grid, recycling a system of grid generators.

The grid inherits the space dimension of the generator system.

Parameters

gs The system of generators defining the grid. Its data-structures may be recycled to build the grid.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 174 of file Grid.inlines.hh.

References construct().

11.71.4.8 template<typename Interval > Parma_Polyhedra_Library::Grid::Grid (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a grid out of a box.

The grid inherits the space dimension of the box. The built grid is the most precise grid that includes the box.

Parameters

box The box representing the grid to be built.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 36 of file Grid.templates.hh.

References con_sys, Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), gen_sys, Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(),

Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_-Library::Congruence_System::insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), max_space_dimension(), Parma_Polyhedra_-Library::neg_assign(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Grid_-Generator::scale_to_divisor(), set_congruences_up_to_date(), set_empty(), set_generators_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::set_sorted(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), throw_space_dimension_overflow(), and Parma_Polyhedra_Library::Grid_Generator_System::unset_pending_rows().

11.71.4.9 template<typename U > Parma_Polyhedra_Library::Grid::Grid (const BD_Shape< U > & *bd*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a grid out of a bounded-difference shape.

The grid inherits the space dimension of the BDS. The built grid is the most precise grid that includes the BDS.

Parameters

bd The BDS representing the grid to be built.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `bd` exceeds the maximum allowed space dimension.

Definition at line 187 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::congruences(), and construct().

11.71.4.10 template<typename U > Parma_Polyhedra_Library::Grid::Grid (const Octagonal_Shape< U > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a grid out of an octagonal shape.

The grid inherits the space dimension of the octagonal shape. The built grid is the most precise grid that includes the octagonal shape.

Parameters

os The octagonal shape representing the grid to be built.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `os` exceeds the maximum allowed space dimension.

Definition at line 202 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::congruences(), and construct().

11.71.4.11 Parma_Polyhedra_Library::Grid::Grid (const Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [explicit]

Builds a grid from a polyhedron using algorithms whose complexity does not exceed the one specified by `complexity`. If `complexity` is `ANY_COMPLEXITY`, then the grid built is the smallest one containing `ph`.

The grid inherits the space dimension of polyhedron.

Parameters

ph The polyhedron.

complexity The complexity class.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 138 of file Grid_public.cc.

References Parma_Polyhedra_Library::Linear_Expression::all_homogeneous_terms_are_zero(), Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Polyhedron::constraints_are_minimized(), Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date(), construct(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Polyhedron::minimize(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, set_empty(), set_zero_dim_univ(), space_dim, and Parma_Polyhedra_Library::Polyhedron::space_dimension().

11.71.4.12 Parma_Polyhedra_Library::Grid::Grid (const Grid & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY)

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 38 of file Grid_public.cc.

References con_sys, congruences_are_up_to_date(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), and space_dim.

11.71.4.13 Parma_Polyhedra_Library::Grid::~Grid () [inline]

Destructor.

Definition at line 216 of file Grid.inlines.hh.

11.71.5 Member Function Documentation

11.71.5.1 void Parma_Polyhedra_Library::Grid::add_congruence (const Congruence & *cg*) [inline]

Adds a copy of congruence `cg` to `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 254 of file Grid.inlines.hh.

References add_congruence_no_check(), marked_empty(), space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), and throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), drop_some_non_integer_points(), refine_with_congruence(), simplify_using_context_assign(), and wrap_assign().

11.71.5.2 void Parma_Polyhedra_Library::Grid::add_congruence_no_check (const Congruence & *cg*) [private]

Adds the congruence `cg` to `*this`.

Warning

If `cg` and `*this` are dimension-incompatible, the grid generator system is not minimized or `*this` is empty, then the behavior is undefined.

Definition at line 647 of file Grid_nonpublic.cc.

References clear_congruences_minimized(), clear_generators_up_to_date(), con_sys, congruences_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), marked_empty(), OK(), set_congruences_up_to_date(), set_empty(), space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), and update_congruences().

Referenced by add_congruence(), add_constraint_no_check(), difference_assign(), generalized_affine_image(), generalized_affine_preimage(), and refine_no_check().

11.71.5.3 void Parma_Polyhedra_Library::Grid::add_congruences (const Congruence_System & *cgs*) [inline]

Adds a copy of each congruence in `cgs` to `*this`.

Parameters

cgs Contains the congruences that will be added to the system of congruences of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 264 of file Grid.inlines.hh.

References add_recycled_congruences(), marked_empty(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), and throw_dimension_incompatible().

Referenced by refine_with_congruences(), and simplify_using_context_assign().

11.71.5.4 void Parma_Polyhedra_Library::Grid::add_constraint (const Constraint & *c*) [inline]

Adds to `*this` a congruence equivalent to constraint `c`.

Parameters

c The constraint to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible or if constraint `c` is not optimally supported by the grid domain.

Definition at line 297 of file Grid.inlines.hh.

References add_constraint_no_check(), marked_empty(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_dimension_incompatible().

Referenced by wrap_assign().

11.71.5.5 void Parma_Polyhedra_Library::Grid::add_constraint_no_check (const Constraint & *c*) [private]

Uses the constraint `c` to refine `*this`.

Parameters

c The constraint to be added.

Exceptions

std::invalid_argument Thrown if c is a non-trivial inequality constraint.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 673 of file Grid_nonpublic.cc.

References add_congruence_no_check(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Constraint::is_tautological(), marked_empty(), set_empty(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_invalid_constraint().

Referenced by add_constraint(), and add_constraints().

11.71.5.6 void Parma_Polyhedra_Library::Grid::add_constraints (const Constraint_System & *cs*)

Adds to `*this` congruences equivalent to the constraints in `cs`.

Parameters

cs The constraints to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible or if `cs` contains a constraint whcih is not optimally supported by the grid domain.

Definition at line 1130 of file Grid_public.cc.

References add_constraint_no_check(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), marked_empty(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), and throw_dimension_incompatible().

Referenced by add_recycled_constraints().

11.71.5.7 void Parma_Polyhedra_Library::Grid::add_grid_generator (const Grid_Generator & *g*)

Adds a copy of grid generator `g` to the system of generators of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible, or if `*this` is an empty grid and `g` is not a point.

Definition at line 1146 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_empty(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter(), Parma_Polyhedra_Library::Grid_Generator::is_parameter(), Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point(), marked_empty(), normalize_divisors(), OK(), set_generators_up_to_date(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Grid_Generator::space_dimension(), throw_dimension_incompatible(), throw_invalid_generator(), and update_generators().

Referenced by generalized_affine_image(), generalized_affine_preimage(), and wrap_assign().

11.71.5.8 void Parma_Polyhedra_Library::Grid::add_grid_generators (const Grid_Generator_System & *gs*)

Adds a copy of the generators in `gs` to the system of generators of `*this`.

Parameters

gs Contains the generators that will be added to the system of generators of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `gs` are dimension-incompatible, or if `*this` is empty and the system of generators `gs` is not empty, but has no points.

Definition at line 1290 of file Grid_public.cc.

References add_recycled_grid_generators().

11.71.5.9 void Parma_Polyhedra_Library::Grid::add_recycled_congruences (Congruence_System & *cgs*)

Adds the congruences in `cgs` to *this.

Parameters

cgs The congruence system to be added to `*this`. The congruences in `cgs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Warning

The only assumption that can be made about `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 1190 of file Grid_public.cc.

References Parma_Polyhedra_Library::Congruence_System::begin(), clear_congruences_minimized(), clear_generators_up_to_date(), con_sys, congruences_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Matrix::has_no_rows(), marked_empty(), OK(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), set_empty(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), throw_dimension_incompatible(), and update_congruences().

Referenced by add_congruences(), congruence_widening_assign(), contains_integer_point(), expand_space_dimension(), generalized_affine_image(), generalized_affine_preimage(), limited_congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), and simplify_using_context_assign().

11.71.5.10 void Parma_Polyhedra_Library::Grid::add_recycled_constraints (Constraint_System & *cs*) `[inline]`

Adds to `*this` congruences equivalent to the constraints in `cs`.

Parameters

cs The constraints to be added. They may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible or if `cs` contains a constraint whcih is not optimally supported by the grid domain.

Warning

The only assumption that can be made about `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 306 of file Grid.inlines.hh.

References add_constraints().

11.71.5.11 void Parma_Polyhedra_Library::Grid::add_recycled_grid_generators (Grid_Generator_System & *gs*)

Adds the generators in `gs` to the system of generators of `this`.

Parameters

gs The generator system to be added to `*this`. The generators in `gs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `gs` are dimension-incompatible.

Warning

The only assumption that can be made about `gs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 1230 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_empty(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Grid_Generator_System::has_points(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), marked_empty(), normalize_divisors(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), set_generators_up_to_date(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), swap(), throw_dimension_incompatible(), throw_invalid_generators(), and update_generators().

Referenced by add_grid_generators(), generalized_affine_image(), generalized_affine_preimage(), and generator_widening_assign().

11.71.5.12 void Parma_Polyhedra_Library::Grid::add_space_dimensions (Grid_Generator_System & *gs*, Congruence_System & *cgs*, dimension_type *dims*) [private]

Adds new space dimensions to the given systems.

Parameters

gs A generator system, to which columns are added;

cgs A congruence system, to which rows and columns are added;

dims The number of space dimensions to add.

This method is invoked only by add_space_dimensions_and_project().

Definition at line 54 of file Grid_chdims.cc.

References Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), dim_kinds, EQUALITY, Parma_Polyhedra_Library::Grid_Generator_System::insert(), normalize_divisors(), Parma_Polyhedra_Library::Matrix::num_columns(), space_dim, and Parma_Polyhedra_Library::Grid_Generator_System::space_dimension().

11.71.5.13 void Parma_Polyhedra_Library::Grid::add_space_dimensions (Congruence_System & *cgs*, Grid_Generator_System & *gs*, dimension_type *dims*) [private]

Adds new space dimensions to the given systems.

Parameters

cgs A congruence system, to which columns are added;

gs A generator system, to which rows and columns are added;

dims The number of space dimensions to add.

This method is invoked only by add_space_dimensions_and_embed().

Definition at line 35 of file Grid_chdims.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::add_universe_rows_and_columns(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), CON_VIRTUAL, congruences_are_minimized(), dim_kinds, generators_are_minimized(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by add_space_dimensions_and_embed(), and add_space_dimensions_and_project().

11.71.5.14 void Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed (dimension_type *m*)

Adds m new space dimensions and embeds the old grid in the new vector space.

Parameters

m The number of dimensions to add.

Exceptions

std::length_error Thrown if adding m new space dimensions would cause the vector space to exceed dimension max_space_dimension().

The new space dimensions will be those having the highest indexes in the new grid, which is characterized by a system of congruences in which the variables which are the new dimensions can have any value. For instance, when starting from the grid $\mathcal{L} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the grid

$$\left\{ (x, y, z)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{L} \right\}.$$

Definition at line 79 of file Grid_chdims.cc.

References add_space_dimensions(), Parma_Polyhedra_Library::Grid_Generator_System::add_universe_rows_and_columns(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), con_sys, CON_VIRTUAL, congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, gen_sys, generators_are_minimized(), generators_are_up_to_date(), LINE, marked_empty(), max_space_dimension(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), set_empty(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), space_dimension(), status, swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Grid::Status::test_zero_dim_univ(), throw_space_dimension_overflow(), and Parma_Polyhedra_Library::UNIVERSE.

Referenced by expand_space_dimension(), generalized_affine_image(), generalized_affine_preimage(), and simplify_using_context_assign().

11.71.5.15 void Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project (dimension_type *m*)

Adds `m` new space dimensions to the grid and does not embed it in the new vector space.

Parameters

m The number of space dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

The new space dimensions will be those having the highest indexes in the new grid, which is characterized by a system of congruences in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the grid $\mathcal{L} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the grid

$$\left\{ (x, y, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{L} \right\}.$$

Definition at line 153 of file Grid_chdims.cc.

References add_space_dimensions(), Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), con_sys, congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, EQUALITY, gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), marked_empty(), max_space_dimension(), normalize_divisors(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), set_empty(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), space_dimension(), status, swap(), Parma_Polyhedra_Library::Grid::Status::test_zero_dim_univ(), throw_space_dimension_overflow(), and Parma_Polyhedra_Library::UNIVERSE.

11.71.5.16 PPL::dimension_type Parma_Polyhedra_Library::Grid::affine_dimension () const

Returns 0, if `*this` is empty; otherwise, returns the affine dimension of `*this`.

Definition at line 258 of file Grid_public.cc.

References con_sys, congruences_are_minimized(), congruences_are_up_to_date(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), is_empty(), minimized_congruences(), minimized_grid_generators(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), and space_dim.

11.71.5.17 void Parma_Polyhedra_Library::Grid::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the affine image of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

When considering the generators of a grid, the affine transformation

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{\text{denominator}}$$

is assigned to `var` where `expr` is $\sum_{i=0}^{n-1} a_i x_i + b$ (b is the inhomogeneous term).

If congruences are up-to-date, it uses the specialized function affine_preimage() (for the system of congruences) and inverse transformation to reach the same result. To obtain the inverse transformation we use the following observation.

Observation:

1. The affine transformation is invertible if the coefficient of `var` in this transformation (i.e., a_{var}) is different from zero.
2. If the transformation is invertible, then we can write

$$\text{denominator} * {x'}_{\text{var}} = \sum_{i=0}^{n-1} a_i x_i + b = a_{\text{var}} x_{\text{var}} + \sum_{i \neq var} a_i x_i + b,$$

so that the inverse transformation is

$$a_{\text{var}} x_{\text{var}} = \text{denominator} * {x'}_{\text{var}} - \sum_{i \neq j} a_i x_i - b.$$

Then, if the transformation is invertible, all the entities that were up-to-date remain up-to-date. Otherwise only generators remain up-to-date.

Definition at line 1757 of file Grid_public.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::affine_image(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), clear_congruences_minimized(), clear_congruences_up_to_date(), clear_generators_minimized(), con_sys, congruences_are_up_to_date(), gen_sys, generators_are_up_to_date(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), normalize_divisors(), OK(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by fold_space_dimensions(), and generalized_affine_image().

11.71.5.18 void Parma_Polyhedra_Library::Grid::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

When considering congruences of a grid, the affine transformation

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{denominator},$$

is assigned to `var` where `expr` is $\sum_{i=0}^{n-1} a_i x_i + b$ (b is the inhomogeneous term).

If generators are up-to-date, then the specialized function affine_image() is used (for the system of generators) and inverse transformation to reach the same result. To obtain the inverse transformation, we use the following observation.

Observation:

1. The affine transformation is invertible if the coefficient of `var` in this transformation (i.e. a_{var}) is different from zero.
2. If the transformation is invertible, then we can write

$$\text{denominator} * {x'}_{\text{var}} = \sum_{i=0}^{n-1} a_i x_i + b = a_{\text{var}} x_{\text{var}} + \sum_{i \neq \text{var}} a_i x_i + b,$$

, the inverse transformation is

$$a_{\text{var}} x_{\text{var}} = \text{denominator} * {x'}_{\text{var}} - \sum_{i \neq j} a_i x_i - b.$$

.

Then, if the transformation is invertible, all the entities that were up-to-date remain up-to-date. Otherwise only congruences remain up-to-date.

Definition at line 1838 of file Grid_public.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::affine_image(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), clear_congruences_minimized(), clear_generators_minimized(), clear_generators_up_to_date(), con_sys, congruences_are_up_to_date(), gen_sys, generators_are_up_to_date(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), OK(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by generalized_affine_preimage().

11.71.5.19 void Parma_Polyhedra_Library::Grid::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 2572 of file Grid_public.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::ascii_dump(), Parma_Polyhedra_Library::Congruence_System::ascii_dump(), Parma_Polyhedra_Library::Grid::Status::ascii_dump(), con_sys, congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, gen_sys, generators_are_minimized(), generators_are_up_to_date(), space_dim, and status.

11.71.5.20 void Parma_Polyhedra_Library::Grid::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by OK().

11.71.5.21 bool Parma_Polyhedra_Library::Grid::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 2602 of file Grid_public.cc.

11.71.5.22 void Parma_Polyhedra_Library::Grid::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2351 of file Grid_public.cc.

References generalized_affine_image(), Parma_Polyhedra_Library::LESS_OR_EQUAL, marked_empty(), OK(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.71.5.23 void Parma_Polyhedra_Library::Grid::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2394 of file Grid_public.cc.

References generalized_affine_preimage(), Parma_Polyhedra_Library::LESS_OR_EQUAL, marked_empty(), OK(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.71.5.24 bool Parma_Polyhedra_Library::Grid::bounds (const Linear_Expression & *expr*, const char * *method_call*) const `[private]`

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

method_call The call description of the public parent method, for example "bounded_from_above(e)". Passed to throw_dimension_incompatible, as the first argument.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 275 of file Grid_nonpublic.cc.

References bounds_no_check(), generators_are_minimized(), generators_are_up_to_date(), marked_empty(), minimize(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by bounds_from_above(), bounds_from_below(), and max_min().

11.71.5.25 bool Parma_Polyhedra_Library::Grid::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded in `*this`.

This method is the same as bounds_from_below.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 312 of file Grid.inlines.hh.

References bounds().

11.71.5.26 bool Parma_Polyhedra_Library::Grid::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded in `*this`.

This method is the same as bounds_from_above.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 317 of file Grid.inlines.hh.

References bounds().

11.71.5.27 bool Parma_Polyhedra_Library::Grid::bounds_no_check (const Linear_Expression & *expr*) const [private]

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

Definition at line 294 of file Grid_nonpublic.cc.

References gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign(), Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), space_dim, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by bounds(), frequency_no_check(), and wrap_assign().

11.71.5.28 bool Parma_Polyhedra_Library::Grid::can_recycle_congruence_systems () [inline, static]

Returns true indicating that this domain has methods that can recycle congruences.

Definition at line 292 of file Grid.inlines.hh.

11.71.5.29 bool Parma_Polyhedra_Library::Grid::can_recycle_constraint_systems () [inline, static]

Returns true indicating that this domain has methods that can recycle constraints.

Definition at line 287 of file Grid.inlines.hh.

11.71.5.30 void Parma_Polyhedra_Library::Grid::clear_congruences_minimized () [inline, private]

Sets `status` to express that congruences are no longer minimized.

Definition at line 87 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::reset_c_minimized(), and status.

Referenced by add_congruence_no_check(), add_recycled_congruences(), affine_image(), affine_preimage(), clear_congruences_up_to_date(), concatenate_assign(), intersection_assign(), and map_space_dimensions().

11.71.5.31 void Parma_Polyhedra_Library::Grid::clear_congruences_up_to_date () [inline, private]

Sets `status` to express that congruences are out of date.

Definition at line 97 of file Grid.inlines.hh.

References clear_congruences_minimized(), Parma_Polyhedra_Library::Grid::Status::reset_c_up_to_date(), and status.

Referenced by add_grid_generator(), add_recycled_grid_generators(), affine_image(), generalized_affine_image(), generalized_affine_preimage(), remove_higher_space_dimensions(), remove_space_dimensions(), time_elapse_assign(), unconstrain(), and upper_bound_assign().

11.71.5.32 void Parma_Polyhedra_Library::Grid::clear_empty () [inline, private]

Clears the `status` flag indicating that the grid is empty.

Definition at line 82 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::reset_empty(), and status.

Referenced by add_grid_generator(), and add_recycled_grid_generators().

11.71.5.33 void Parma_Polyhedra_Library::Grid::clear_generators_minimized () [inline, private]

Sets `status` to express that generators are no longer minimized.

Definition at line 92 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::reset_g_minimized(), and status.

Referenced by add_grid_generator(), add_recycled_grid_generators(), affine_image(), affine_preimage(), clear_generators_up_to_date(), generalized_affine_image(), generalized_affine_preimage(), map_space_dimensions(), remove_space_dimensions(), time_elapse_assign(), unconstrain(), and upper_bound_assign().

11.71.5.34 void Parma_Polyhedra_Library::Grid::clear_generators_up_to_date () [inline, private]

Sets `status` to express that generators are out of date.

Definition at line 104 of file Grid.inlines.hh.

References clear_generators_minimized(), Parma_Polyhedra_Library::Grid::Status::reset_g_up_to_date(), and status.

Referenced by add_congruence_no_check(), add_recycled_congruences(), affine_preimage(), concatenate_assign(), intersection_assign(), OK(), and remove_higher_space_dimensions().

11.71.5.35 void Parma_Polyhedra_Library::Grid::concatenate_assign (const Grid & *y*)

Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

Exceptions

std::length_error Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 218 of file Grid_chdims.cc.

References clear_congruences_minimized(), clear_generators_up_to_date(), con_sys, Parma_Polyhedra_Library::Congruence_System::concatenate(), congruences(), congruences_are_up_to_date(), marked_empty(), max_space_dimension(), OK(), set_empty(), space_dim, space_dimension(), throw_space_dimension_overflow(), and update_congruences().

11.71.5.36 void Parma_Polyhedra_Library::Grid::congruence_widening_assign (const Grid & *y*, unsigned * *tp* = NULL)

Assigns to `*this` the result of computing the Grid widening between `*this` and `y` using congruence systems.

Parameters

y A grid that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 76 of file Grid_widenings.cc.

References add_recycled_congruences(), con_sys, congruences_are_minimized(), congruences_are_up_to_date(), contains(), dim_kinds, marked_empty(), Parma_Polyhedra_Library::Congruence_System::num_equalities(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), select_wider_congruences(), set_congruences_minimized(), set_empty(), simplify(), space_dim, swap(), throw_dimension_incompatible(), and update_congruences().

Referenced by limited_congruence_extrapolation_assign(), and widening_assign().

11.71.5.37 const PPL::Congruence_System & Parma_Polyhedra_Library::Grid::congruences () const

Returns the system of congruences.

Definition at line 279 of file Grid_public.cc.

References con_sys, congruences_are_up_to_date(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), space_dim, and update_congruences().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), concatenate_assign(), constraints(), difference_assign(), expand_space_dimension(), and minimized_congruences().

11.71.5.38 bool Parma_Polyhedra_Library::Grid::congruences_are_minimized () const [inline, private]

Returns `true` if the system of congruences is minimized.

Definition at line 50 of file Grid.inlines.hh.

References status, and Parma_Polyhedra_Library::Grid::Status::test_c_minimized().

Referenced by add_space_dimensions(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_dimension(), ascii_dump(), congruence_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), is_empty(), is_included_in(), is_universe(), minimize(),

minimized_congruences(), OK(), quick_equivalence_test(), remove_higher_space_dimensions(), select_-wider_congruences(), simplify_using_context_assign(), and update_generators().

11.71.5.39 bool Parma_Polyhedra_Library::Grid::congruences_are_up_to_date () const [inline, private]

Returns `true` if the system of congruences is up-to-date.

Definition at line 40 of file Grid.inlines.hh.

References status, and Parma_Polyhedra_Library::Grid::Status::test_c_up_to_date().

Referenced by add_congruence_no_check(), add_recycled_congruences(), add_space_dimensions_and_-embed(), add_space_dimensions_and_project(), affine_dimension(), affine_image(), affine_preimage(), ascii_dump(), concatenate_assign(), congruence_widening_assign(), congruences(), constrains(), Grid(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), intersection_assign(), is_included_in(), is_universe(), map_space_dimensions(), minimize(), minimized_congruences(), OK(), operator=(), relation_with(), simplify_using_context_assign(), update_generators(), and widening_assign().

11.71.5.40 bool Parma_Polyhedra_Library::Grid::constrains (Variable *var*) const

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 800 of file Grid_public.cc.

References Parma_Polyhedra_Library::Grid_Generator::coefficient(), con_sys, congruences_are_up_-to_date(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_-Library::Variable::id(), Parma_Polyhedra_Library::Grid_Generator::is_line(), marked_empty(), mini-mize(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Grid_Generator_-System::num_rows(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), throw_-dimension_incompatible(), and update_congruences().

11.71.5.41 Constraint_System Parma_Polyhedra_Library::Grid::constraints () const [inline]

Returns a system of equality constraints satisfied by `*this` with the same affine dimension as `*this`.

Definition at line 235 of file Grid.inlines.hh.

References congruences().

Referenced by Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), and Parma_Polyhedra_-Library::NNC_Polyhedron::NNC_Polyhedron().

11.71.5.42 void Parma_Polyhedra_Library::Grid::construct (Grid_Generator_System & *ggs*) [private]

Builds a grid from a system of grid generators.

The grid inherits the space dimension of the generator system.

Parameters

ggs The system of grid generators defining the grid. Its data-structures may be recycled to build the grid.

Definition at line 143 of file Grid_nonpublic.cc.

References con_sys, gen_sys, Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Grid_Generator_System::has_points(), Parma_Polyhedra_Library::Congruence_System::insert(), max_space_dimension(), normalize_divisors(), OK(), Parma_Polyhedra_Library::Grid::Status::set_empty(), set_generators_up_to_date(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), status, swap(), throw_invalid_generators(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

11.71.5.43 void Parma_Polyhedra_Library::Grid::construct (Congruence_System & *cgs*) [private]

Builds a grid from a system of congruences.

The grid inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the grid. Its data-structures may be recycled to build the grid.

Definition at line 105 of file Grid_nonpublic.cc.

References con_sys, gen_sys, Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Congruence_System::insert(), max_space_dimension(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), set_congruences_up_to_date(), Parma_Polyhedra_Library::Grid::Status::set_empty(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), Parma_Polyhedra_Library::Congruence_System::space_dimension(), status, swap(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

11.71.5.44 void Parma_Polyhedra_Library::Grid::construct (dimension_type *num_dimensions*, Degenerate_Element *kind*) [private]

Builds a grid universe or empty grid.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the grid;

kind specifies whether the universe or the empty grid has to be built.

Definition at line 52 of file Grid_nonpublic.cc.

References con_sys, CON_VIRTUAL, dim_kinds, Parma_Polyhedra_Library::EMPTY, gen_sys, Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), OK(), PROPER_CONGRUENCE, set_congruences_minimized(), Parma_Polyhedra_Library::Grid::Status::set_empty(), set_generators_minimized(), set_zero_dim_univ(), space_dim, status, Parma_Polyhedra_Library::Congruence_System::swap(), swap(), Parma_Polyhedra_Library::Congruence::zero_dim_false(), and Parma_Polyhedra_Library::Congruence::zero_dim_integrality().

Referenced by Grid().

11.71.5.45 bool Parma_Polyhedra_Library::Grid::contains (const Grid & *y*) const

Returns `true` if and only if `*this` contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2543 of file Grid_public.cc.

References is_empty(), is_included_in(), marked_empty(), quick_equivalence_test(), space_dim, throw_dimension_incompatible(), and TVB_TRUE.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), congruence_widening_assign(), difference_assign(), generator_widening_assign(), limited_congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), strictly_contains(), and widening_assign().

11.71.5.46 bool Parma_Polyhedra_Library::Grid::contains_integer_point () const

Returns `true` if and only if `*this` contains at least one integer point.

Definition at line 778 of file Grid_public.cc.

References add_recycled_congruences(), Parma_Polyhedra_Library::Congruence_System::insert(), is_empty(), marked_empty(), and space_dim.

11.71.5.47 void Parma_Polyhedra_Library::Grid::conversion (Grid_Generator_System & *source*, Congruence_System & *dest*, Dimension_Kinds & *dim_kinds*) [static, private]

Converts congruence system `dest` to be equivalent to generator system `source`.

Definition at line 155 of file Grid_conversion.cc.

References CON_VIRTUAL, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), GEN_VIRTUAL, Parma_Polyhedra_Library::lcm_assign(), LINE, lower_triangular(), multiply_grid(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), PARAMETER, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Congruence_System::resize_no_copy(), Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), Parma_Polyhedra_Library::sub_mul_assign(), and upper_triangular().

11.71.5.48 void Parma_Polyhedra_Library::Grid::conversion (Congruence_System & *source*, Grid_Generator_System & *dest*, Dimension_Kinds & *dim_kinds*) [static, private]

Converts generator system `dest` to be equivalent to congruence system `source`.

Definition at line 315 of file Grid_conversion.cc.

References CON_VIRTUAL, EQUALITY, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), GEN_VIRTUAL, Parma_Polyhedra_Library::lcm_assign(), LINE, lower_triangular(), multiply_grid(), Parma_Polyhedra_Library::Matrix::num_columns(), PARAMETER, PPL_DIRTY_TEMP_COEFFICIENT, PROPER_CONGRUENCE, Parma_Polyhedra_Library::Grid_Generator_System::resize_no_copy(), Parma_Polyhedra_Library::Grid_Generator_System::set_index_first_pending_row(), Parma_Polyhedra_Library::Grid_Generator::set_is_line(), Parma_Polyhedra_Library::Grid_Generator::set_is_parameter_or_point(), Parma_Polyhedra_Library::sub_mul_assign(), and upper_triangular().

Referenced by update_generators().

11.71.5.49 void Parma_Polyhedra_Library::Grid::difference_assign (const Grid & *y*)

Assigns to `*this` the grid-difference of `*this` and `y`.

The grid difference between grids x and y is the smallest grid containing all the points from x and y that are only in x.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1484 of file Grid_public.cc.

References add_congruence_no_check(), Parma_Polyhedra_Library::Congruence_System::begin(), congruences(), contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), marked_empty(), Parma_Polyhedra_Library::Congruence::modulus(), OK(), relation_with(), set_empty(), space_dim, throw_dimension_incompatible(), and upper_bound_assign().

Referenced by upper_bound_assign_if_exact().

11.71.5.50 void Parma_Polyhedra_Library::Grid::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY)

Possibly tightens `*this` by dropping all points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 2872 of file Grid_public.cc.

References add_congruence(), marked_empty(), OK(), space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), and throw_dimension_incompatible().

11.71.5.51 void Parma_Polyhedra_Library::Grid::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY)

Possibly tightens `*this` by dropping all points with non-integer coordinates.

Parameters

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 2857 of file Grid_public.cc.

References add_congruence(), marked_empty(), OK(), and space_dim.

11.71.5.52 void Parma_Polyhedra_Library::Grid::expand_space_dimension (Variable *var*, dimension_type *m*)

Creates `m` copies of the space dimension corresponding to `var`.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if `var` does not correspond to a dimension of the vector space.

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

If `*this` has space dimension n, with $n > 0$, and `var` has space dimension $k \leq n$, then the k-th space dimension is expanded to `m` new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 395 of file Grid_chdims.cc.

References add_recycled_congruences(), add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence::coefficient(), congruences(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), max_space_dimension(), Parma_Polyhedra_Library::Congruence::modulus(), OK(), space_dim, space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and throw_space_dimension_overflow().

11.71.5.53 PPL::memory_size_type Parma_Polyhedra_Library::Grid::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Definition at line 2670 of file Grid_public.cc.

References con_sys, Parma_Polyhedra_Library::Grid_Generator_System::external_memory_in_bytes(), Parma_Polyhedra_Library::Congruence_System::external_memory_in_bytes(), and gen_sys.

Referenced by total_memory_in_bytes().

11.71.5.54 void Parma_Polyhedra_Library::Grid::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*)

Folds the space dimensions in `vars` into `dest`.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 445 of file Grid_chdims.cc.

References affine_image(), grid_generators(), Parma_Polyhedra_Library::Variable::id(), marked_empty(), OK(), remove_space_dimensions(), space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), throw_invalid_argument(), and upper_bound_assign().

11.71.5.55 bool Parma_Polyhedra_Library::Grid::frequency (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const

Returns `true` if and only if `*this` is not empty and frequency for `*this` with respect to `expr` is defined, in which case the frequency and the value for `expr` that is closest to zero are computed.

Parameters

expr The linear expression for which the frequency is needed;

freq_n The numerator of the maximum frequency of `expr`;

freq_d The denominator of the maximum frequency of `expr`;

val_n The numerator of them value of `expr` at a point in the grid that is closest to zero;

val_d The denominator of a value of `expr` at a point in the grid that is closest to zero;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If *this is empty or frequency is undefined with respect to expr, then false is returned and freq_n, freq_d, val_n and val_d are left untouched.

Definition at line 2487 of file Grid_public.cc.

References frequency_no_check(), generators_are_minimized(), is_empty(), minimize(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and throw_dimension_incompatible().

11.71.5.56 bool Parma_Polyhedra_Library::Grid::frequency_no_check (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const [private]

Returns true if and only if *this is not empty and frequency for *this with respect to expr is defined, in which case the frequency and the value for expr that is closest to zero are computed.

Parameters

expr The linear expression for which the frequency is needed;

freq_n The numerator of the maximum frequency of expr;

freq_d The denominator of the maximum frequency of expr;

val_n The numerator of a value of expr at a point in the grid that is closest to zero;

val_d The denominator of a value of expr at a point in the grid that is closest to zero;

If *this is empty or frequency is undefined with respect to expr, then false is returned and freq_n, freq_d, val_n and val_d are left untouched.

Warning

If expr and *this are dimension-incompatible, the grid generator system is not minimized or *this is empty, then the behavior is undefined.

Definition at line 315 of file Grid_nonpublic.cc.

References bounds_no_check(), Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Grid_Generator::is_line(), Parma_Polyhedra_Library::Grid_Generator::is_parameter(), Parma_Polyhedra_Library::Grid_Generator::is_point(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, space_dim, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by frequency(), and wrap_assign().

11.71.5.57 void Parma_Polyhedra_Library::Grid::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*, Coefficient_traits::const_reference *modulus* = Coefficient_zero())

Assigns to *this the image of *this with respect to the generalized affine relation $\mathrm{lhs}' = \mathrm{rhs} \pmod{\mathrm{modulus}}$.

Parameters

lhs The left hand side affine expression.

relsym The relation symbol where EQUAL is the symbol for a congruence relation;

rhs The right hand side affine expression.

modulus The modulus of the congruence lhs = rhs. A modulus of zero indicates lhs == rhs. Optional argument with an automatic value of zero.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs`.

Definition at line 2086 of file Grid_public.cc.

References add_congruence_no_check(), add_grid_generator(), add_recycled_congruences(), add_recycled_grid_generators(), add_space_dimensions_and_embed(), clear_congruences_up_to_date(), clear_generators_minimized(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), is_empty(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), normalize_divisors(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), remove_higher_space_dimensions(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), throw_invalid_argument(), and update_generators().

11.71.5.58 void Parma_Polyhedra_Library::Grid::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`, Coefficient_traits::const_reference *modulus* = `Coefficient_zero()`)

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{var}' = \frac{\mathrm{expr}}{\mathrm{denominator}} \pmod{\mathrm{modulus}}$.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol where EQUAL is the symbol for a congruence relation;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression. Optional argument with an automatic value of one;

modulus The modulus of the congruence lhs = rhs. A modulus of zero indicates lhs == rhs. Optional argument with an automatic value of zero.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `this`.

Definition at line 1912 of file Grid_public.cc.

References add_grid_generator(), affine_image(), clear_congruences_up_to_date(), clear_generators_minimized(), Parma_Polyhedra_Library::EQUAL, gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), marked_empty(), minimize(), normalize_divisors(),

OK(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_-Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_-argument().

Referenced by bounded_affine_image(), and generalized_affine_preimage().

11.71.5.59 void Parma_Polyhedra_Library::Grid::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*, Coefficient_traits::const_reference *modulus* = `Coefficient_zero()`)

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{lhs}' = \mathrm{rhs} \pmod{\mathrm{modulus}}$.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol where EQUAL is the symbol for a congruence relation;

rhs The right hand side affine expression;

modulus The modulus of the congruence lhs = rhs. A modulus of zero indicates lhs == rhs. Optional argument with an automatic value of zero.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs`.

Definition at line 2220 of file Grid_public.cc.

References add_congruence_no_check(), add_grid_generator(), add_recycled_congruences(), add_-recycled_grid_generators(), add_space_dimensions_and_embed(), clear_congruences_up_to_date(), clear_generators_minimized(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_-Polyhedra_Library::EQUAL, gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_-Generator_System::insert(), is_empty(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_-assign(), normalize_divisors(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_-Library::Grid_Generator_System::recycling_insert(), remove_higher_space_dimensions(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), throw_invalid_argument(), and update_generators().

11.71.5.60 void Parma_Polyhedra_Library::Grid::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`, Coefficient_traits::const_reference *modulus* = `Coefficient_zero()`)

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{var}' = \frac{\mathrm{expr}}{\mathrm{denominator}} \pmod{\mathrm{modulus}}$.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol where EQUAL is the symbol for a congruence relation;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression. Optional argument with an automatic value of one;

modulus The modulus of the congruence lhs = rhs. A modulus of zero indicates lhs == rhs. Optional argument with an automatic value of zero.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `this`.

Definition at line 1993 of file Grid_public.cc.

References add_congruence_no_check(), add_grid_generator(), affine_preimage(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, generalized_affine_image(), generators_are_up_to_date(), is_empty(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by bounded_affine_preimage().

11.71.5.61 void Parma_Polyhedra_Library::Grid::generator_widening_assign (const Grid & *y*, unsigned * *tp* = NULL)

Assigns to `*this` the result of computing the Grid widening between `*this` and `y` using generator systems.

Parameters

y A grid that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 278 of file Grid_widenings.cc.

References add_recycled_grid_generators(), contains(), dim_kinds, Parma_Polyhedra_Library::EMPTY, gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_lines(), Parma_Polyhedra_Library::Grid_Generator_System::num_parameters(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), OK(), select_wider_generators(), set_generators_minimized(), simplify(), space_dim, swap(), throw_dimension_incompatible(), and update_generators().

Referenced by limited_generator_extrapolation_assign(), and widening_assign().

11.71.5.62 bool Parma_Polyhedra_Library::Grid::generators_are_minimized () const [inline, private]

Returns `true` if the system of generators is minimized.

Definition at line 55 of file Grid.inlines.hh.

References status, and Parma_Polyhedra_Library::Grid::Status::test_g_minimized().

Referenced by add_space_dimensions(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_dimension(), ascii_dump(), bounds(), bounds_no_check(), constrains(), frequency(), frequency_no_check(), generator_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), max_min(), minimize(), minimized_grid_generators(), OK(), quick_equivalence_test(), remove_higher_space_dimensions(), select_wider_generators(), simplify_using_context_assign(), update_congruences(), and wrap_assign().

11.71.5.63 bool Parma_Polyhedra_Library::Grid::generators_are_up_to_date () const [inline, private]

Returns `true` if the system of generators is up-to-date.

Definition at line 45 of file Grid.inlines.hh.

References status, and Parma_Polyhedra_Library::Grid::Status::test_g_up_to_date().

Referenced by add_grid_generator(), add_recycled_grid_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_dimension(), affine_image(), affine_preimage(), ascii_dump(), bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), constrains(), generalized_affine_image(), generalized_affine_preimage(), generator_widening_assign(), Grid(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), grid_generators(), is_bounded(), is_discrete(), is_empty(), limited_congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), map_space_dimensions(), minimize(), minimized_grid_generators(), OK(), operator=(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), time_elapse_assign(), unconstrain(), upper_bound_assign(), upper_bound_assign_if_exact(), and widening_assign().

11.71.5.64 const PPL::Grid_Generator_System & Parma_Polyhedra_Library::Grid::grid_generators () const

Returns the system of generators.

Definition at line 309 of file Grid_public.cc.

References gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), and update_generators().

Referenced by fold_space_dimensions(), and map_space_dimensions().

11.71.5.65 int32_t Parma_Polyhedra_Library::Grid::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 230 of file Grid.inlines.hh.

References space_dimension().

11.71.5.66 void Parma_Polyhedra_Library::Grid::intersection_assign (const Grid & *y*)

Assigns to `*this` the intersection of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1372 of file Grid_public.cc.

References clear_congruences_minimized(), clear_generators_up_to_date(), con_sys, congruences_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Congruence_System::insert(), marked_empty(), OK(), set_empty(), space_dim, throw_dimension_incompatible(), and update_congruences().

Referenced by is_disjoint_from().

11.71.5.67 bool Parma_Polyhedra_Library::Grid::is_bounded () const

Returns `true` if and only if `*this` is bounded.

Definition at line 731 of file Grid_public.cc.

References gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), space_dim, and update_generators().

11.71.5.68 bool Parma_Polyhedra_Library::Grid::is_discrete () const

Returns `true` if and only if `*this` is discrete.

A grid is discrete if it can be defined by a generator system which contains only points and parameters. This includes the empty grid and any grid in dimension zero.

Definition at line 755 of file Grid_public.cc.

References gen_sys, generators_are_up_to_date(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), space_dim, and update_generators().

11.71.5.69 bool Parma_Polyhedra_Library::Grid::is_disjoint_from (const Grid & *y*) const

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are dimension-incompatible.

Definition at line 2562 of file Grid_public.cc.

References intersection_assign(), is_empty(), space_dim, and throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment().

11.71.5.70 bool Parma_Polyhedra_Library::Grid::is_empty () const

Returns `true` if and only if `*this` is an empty grid.

Definition at line 678 of file Grid_public.cc.

References con_sys, congruences_are_minimized(), dim_kinds, generators_are_up_to_date(), marked_empty(), set_congruences_minimized(), set_empty(), simplify(), and space_dim.

Referenced by affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), contains(), contains_integer_point(), frequency(), generalized_affine_image(), generalized_affine_preimage(), is_disjoint_from(), operator<<(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), remove_higher_space_dimensions(), and simplify_using_context_assign().

11.71.5.71 bool Parma_Polyhedra_Library::Grid::is_included_in (const Grid & *y*) const [private]

Returns `true` if and only if `*this` is included in `y`.

Definition at line 238 of file Grid_nonpublic.cc.

References con_sys, congruences_are_minimized(), congruences_are_up_to_date(), marked_empty(), minimize(), OK(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), space_dim, and update_congruences().

Referenced by contains(), simplify_using_context_assign(), and upper_bound_assign_if_exact().

11.71.5.72 bool Parma_Polyhedra_Library::Grid::is_topologically_closed () const

Returns `true` if and only if `*this` is a topologically closed subset of the vector space.

A grid is always topologically closed.

Definition at line 773 of file Grid_public.cc.

11.71.5.73 bool Parma_Polyhedra_Library::Grid::is_universe () const

Returns `true` if and only if `*this` is a universe grid.

Definition at line 701 of file Grid_public.cc.

References con_sys, congruences_are_minimized(), congruences_are_up_to_date(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Congruence_-

System::satisfies_all_congruences(), space_dim, and update_congruences().

Referenced by operator<<().

11.71.5.74 void Parma_Polyhedra_Library::Grid::limited_congruence_extrapolation_assign (const Grid & *y*, const Congruence_System & *cgs*, unsigned * *tp* = NULL)

Improves the result of the congruence variant of Grid widening computation by also enforcing those congruences in `cgs` that are satisfied by all the points of `*this`.

Parameters

y A grid that *must* be contained in `*this`;

cgs The system of congruences used to improve the widened grid;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cgs` are dimension-incompatible.

Definition at line 160 of file Grid_widenings.cc.

References add_recycled_congruences(), congruence_widening_assign(), contains(), generators_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), throw_dimension_incompatible(), update_generators(), and widening_assign().

11.71.5.75 void Parma_Polyhedra_Library::Grid::limited_extrapolation_assign (const Grid & *y*, const Congruence_System & *cgs*, unsigned * *tp* = NULL)

Improves the result of the Grid widening computation by also enforcing those congruences in `cgs` that are satisfied by all the points of `*this`.

Parameters

y A grid that *must* be contained in `*this`;

cgs The system of congruences used to improve the widened grid;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cgs` are dimension-incompatible.

Definition at line 468 of file Grid_widenings.cc.

References add_recycled_congruences(), contains(), generators_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), throw_dimension_incompatible(), update_generators(), and widening_assign().

11.71.5.76 void Parma_Polyhedra_Library::Grid::limited_generator_extrapolation_assign (const Grid & *y*, const Congruence_System & *cgs*, unsigned * *tp* = NULL)

Improves the result of the generator variant of the Grid widening computation by also enforcing those congruences in `cgs` that are satisfied by all the points of `*this`.

Parameters

y A grid that *must* be contained in `*this`;

cgs The system of congruences used to improve the widened grid;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cgs` are dimension-incompatible.

Definition at line 362 of file Grid_widenings.cc.

References add_recycled_congruences(), contains(), generator_widening_assign(), generators_are_up_-to_date(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Poly_-Con_Relation::is_included(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), throw_-dimension_incompatible(), and update_generators().

11.71.5.77 bool Parma_Polyhedra_Library::Grid::lower_triangular (const Congruence_System & *sys*, const Dimension_Kinds & *dim_kinds*) [static, private]

If `sys` is lower triangular return `true`, else return `false`.

Definition at line 38 of file Grid_conversion.cc.

References CON_VIRTUAL, Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_-Polyhedra_Library::Matrix::num_rows().

Referenced by conversion(), and OK().

11.71.5.78 template<typename Partial_Function > void Parma_Polyhedra_-Library::Grid::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

If `pfunc` maps only some of the dimensions of `*this` then the rest will be projected away.

If the highest dimension mapped to by `pfunc` is higher than the highest dimension in `*this` then the number of dimensions in `this` will be increased to the highest dimension mapped to by `pfunc`.

Parameters

pfunc The partial function specifying the destiny of each space dimension.

The template type parameter Partial_Function must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty codomain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the codomain of the partial function. The `max_in_codomain()` method is called at most once.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned. This method is called at most n times, where n is the dimension of the vector space enclosing the grid.

The result is undefined if `pfunc` does not encode a partial function with the properties described in the specification of the mapping operator.

Definition at line 112 of file Grid.templates.hh.

References Parma_Polyhedra_Library::Grid_Generator_System::begin(), clear_congruences_minimized(), clear_generators_minimized(), Parma_Polyhedra_Library::Grid_Generator::coefficient(), con_sys, congruences_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Grid_Generator_System::end(), gen_sys, generators_are_up_to_date(), grid_generators(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Grid_Generator::LINE, marked_empty(), Parma_Polyhedra_Library::not_a_dimension(), OK(), Parma_Polyhedra_Library::Grid_Generator::PARAMETER, Parma_Polyhedra_Library::Grid_Generator_System::permute_columns(), Parma_Polyhedra_Library::Matrix::permute_columns(), Parma_Polyhedra_Library::Grid_Generator::POINT, set_empty(), Parma_Polyhedra_Library::Grid_Generator_System::set_sorted(), set_zero_dim_univ(), space_dim, swap(), throw_invalid_argument(), Parma_Polyhedra_Library::Grid_Generator::type(), and update_generators().

11.71.5.79 bool Parma_Polyhedra_Library::Grid::marked_empty () const [inline, private]

Returns `true` if the grid is known to be empty.

The return value `false` does not necessarily implies that `*this` is non-empty.

Definition at line 35 of file Grid.inlines.hh.

References status, and Parma_Polyhedra_Library::Grid::Status::test_empty().

Referenced by add_congruence(), add_congruence_no_check(), add_congruences(), add_constraint(), add_constraint_no_check(), add_constraints(), add_grid_generator(), add_recycled_congruences(), add_recycled_grid_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), bounded_affine_image(), bounded_affine_preimage(), bounds(), bounds_no_check(), Parma_Polyhedra_Library::Box< ITV >::Box(), concatenate_assign(), congruence_widening_assign(), congruences(), constrains(), contains(), contains_integer_point(), difference_assign(), drop_some_non_integer_points(), fold_space_dimensions(), frequency_no_check(), generalized_affine_image(), generalized_affine_preimage(), generator_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), grid_generators(), intersection_assign(), is_bounded(), is_discrete(),

is_empty(), is_included_in(), is_universe(), limited_congruence_extrapolation_assign(), limited_-extrapolation_assign(), limited_generator_extrapolation_assign(), map_space_dimensions(), max_min(), minimize(), minimized_grid_generators(), OK(), operator=(), quick_equivalence_test(), refine_no_-check(), refine_with_constraint(), refine_with_constraints(), relation_with(), remove_space_dimensions(), select_wider_congruences(), select_wider_generators(), time_elapse_assign(), unconstrain(), update_-congruences(), update_generators(), upper_bound_assign(), upper_bound_assign_if_exact(), and wrap_-assign().

11.71.5.80 bool Parma_Polyhedra_Library::Grid::max_min (const Linear_Expression & *expr*, const char * *method_call*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*, Generator * *point* = NULL) const [private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

method_call The call description of the public parent method, for example "maximize(e)". Passed to throw_dimension_incompatible, as the first argument;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` in `this` can actually be reached (which is always the case);

point When maximization or minimization succeeds, will be assigned the point where `expr` reaches the extremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `point` are left untouched.

Definition at line 394 of file Grid_nonpublic.cc.

References bounds(), Parma_Polyhedra_Library::Grid_Generator::coefficient(), dim_kinds, Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::exact_div_-assign(), Parma_Polyhedra_Library::gcd_assign(), gen_sys, generators_are_minimized(), Parma_-Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Linear_-Expression::inhomogeneous_term(), marked_empty(), Parma_Polyhedra_Library::Generator::point(), PPL_DIRTY_TEMP_COEFFICIENT, set_generators_minimized(), simplify(), and space_dim.

Referenced by maximize(), and minimize().

11.71.5.81 dimension_type Parma_Polyhedra_Library::Grid::max_space_dimension () [inline, static]

Returns the maximum space dimension all kinds of Grid can handle.

Definition at line 111 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid_Generator_System::max_space_dimension(), and Parma_Polyhedra_Library::Congruence_System::max_space_dimension().

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), concatenate_assign(), construct(), expand_space_dimension(), and Grid().

11.71.5.82 bool Parma_Polyhedra_Library::Grid::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *point*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if the supremum value can be reached in `this`. Always `true` when `this` bounds `expr`. Present for interface compatibility with class Polyhedron, where closure points can result in a value of false;

point When maximization succeeds, will be assigned a point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded by `*this`, `false` is returned and `sup_n`, `sup_d`, `maximum` and `point` are left untouched.

Definition at line 328 of file Grid.inlines.hh.

References max_min().

11.71.5.83 bool Parma_Polyhedra_Library::Grid::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if the supremum value can be reached in `this`. Always `true` when `this` bounds `expr`. Present for interface compatibility with class Polyhedron, where closure points can result in a value of false.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded by `*this`, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 322 of file Grid.inlines.hh.

References max_min().

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box().

11.71.5.84 bool Parma_Polyhedra_Library::Grid::minimize () const [private]

Minimizes both the congruences and the generators.

Returns

`false` if and only if `*this` turns out to be an empty grid.

Minimization is performed on each system only if the minimized Status field is clear.

Definition at line 516 of file Grid_nonpublic.cc.

References con_sys, congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, gen_sys, generators_are_minimized(), generators_are_up_to_date(), marked_empty(), OK(), set_congruences_minimized(), set_generators_minimized(), simplify(), space_dim, update_congruences(), and update_generators().

Referenced by affine_image(), affine_preimage(), bounds(), constrains(), frequency(), generalized_affine_image(), generalized_affine_preimage(), and wrap_assign().

11.71.5.85 bool Parma_Polyhedra_Library::Grid::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *point*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if the is the infimum value can be reached in `this`. Always `true` when `this` bounds `expr`. Present for interface compatibility with class Polyhedron, where closure points can result in a value of false;

point When minimization succeeds, will be assigned a point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If *this is empty or expr is not bounded from below, false is returned and inf_n, inf_d, minimum and point are left untouched.

Definition at line 341 of file Grid.inlines.hh.

References max_min().

11.71.5.86 bool Parma_Polyhedra_Library::Grid::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns true if and only if *this is not empty and expr is bounded from below in *this, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to *this;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum true if the is the infimum value can be reached in this. Always true when this bounds expr. Present for interface compatibility with class Polyhedron, where closure points can result in a value of false.

Exceptions

std::invalid_argument Thrown if expr and *this are dimension-incompatible.

If *this is empty or expr is not bounded from below, false is returned and inf_n, inf_d and minimum are left untouched.

Definition at line 335 of file Grid.inlines.hh.

References max_min().

Referenced by is_included_in(), and simplify_using_context_assign().

11.71.5.87 const PPL::Congruence_System & Parma_Polyhedra_Library::Grid::minimized_congruences () const

Returns the system of congruences in minimal form.

Definition at line 296 of file Grid_public.cc.

References con_sys, congruences(), congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, set_congruences_minimized(), set_empty(), and simplify().

Referenced by affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), minimized_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and operator<<().

11.71.5.88 Constraint_System Parma_Polyhedra_Library::Grid::minimized_constraints () const [inline]

Returns a minimal system of equality constraints satisfied by `*this` with the same affine dimension as `*this`.

Definition at line 240 of file Grid.inlines.hh.

References minimized_congruences().

11.71.5.89 const PPL::Grid_Generator_System & Parma_Polyhedra_Library::Grid::minimized_grid_generators () const

Returns the minimized system of generators.

Definition at line 331 of file Grid_public.cc.

References dim_kinds, gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), marked_empty(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), set_generators_minimized(), simplify(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), and update_generators().

Referenced by affine_dimension().

11.71.5.90 void Parma_Polyhedra_Library::Grid::multiply_grid (const Coefficient & *multiplier*, Grid_Generator & *gen*, Grid_Generator_System & *dest*, dimension_type *num_rows*, dimension_type *num_dims*) [static, private]

Multiply the elements of `dest` by `multiplier`.

Definition at line 105 of file Grid_conversion.cc.

References Parma_Polyhedra_Library::Grid_Generator::is_line(), and Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point().

11.71.5.91 void Parma_Polyhedra_Library::Grid::multiply_grid (const Coefficient & *multiplier*, Congruence & *cg*, Congruence_System & *dest*, dimension_type *num_rows*, dimension_type *num_dims*) [static, private]

Multiply the elements of `dest` by `multiplier`.

Definition at line 128 of file Grid_conversion.cc.

References Parma_Polyhedra_Library::Congruence::is_equality(), and Parma_Polyhedra_Library::Congruence::is_proper_congruence().

Referenced by conversion().

11.71.5.92 void Parma_Polyhedra_Library::Grid::normalize_divisors (Grid_Generator_System & *sys*, Grid_Generator_System & *gen_sys*) [static, private]

Normalize all the divisors in `sys` and `gen_sys`.

Modify sys and gen_sys to use the same single divisor value for all generators, leaving each system representing the grid it represented originally.

Parameters

sys The first of the generator systems to be normalized.

gen_sys The second of the generator systems to be normalized. This system must have at least one row and the divisors of the generators in this system must be equal.

Exceptions

std::runtime_error Thrown if all rows in gen_sys are lines and/or parameters.

Definition at line 571 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point(), normalize_divisors(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), and PPL_DIRTY_TEMP_COEFFICIENT.

11.71.5.93 void Parma_Polyhedra_Library::Grid::normalize_divisors (Grid_Generator_System & *sys*) [inline, static, private]

Normalizes the divisors in sys.

Converts sys to an equivalent system in which the divisors are of equal value.

Parameters

sys The generator system to be normalized. It must have at least one row.

Definition at line 348 of file Grid.inlines.hh.

References normalize_divisors(), and PPL_DIRTY_TEMP_COEFFICIENT.

11.71.5.94 void Parma_Polyhedra_Library::Grid::normalize_divisors (Grid_Generator_System & *sys*, Coefficient & *divisor*, const Grid_Generator * *first_point* = NULL) [static, private]

Normalizes the divisors in sys.

Converts sys to an equivalent system in which the divisors are of equal value.

Parameters

sys The generator system to be normalized. It must have at least one row.

divisor A reference to the initial value of the divisor. The resulting value of this object is the new system divisor.

first_point If first_point has a value other than NULL then it is taken as the first point in sys, and it is assumed that any following points have the same divisor as first_point.

Definition at line 611 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Grid_Generator::divisor(), Parma_Polyhedra_Library::Grid_-Generator::is_parameter_or_point(), Parma_Polyhedra_Library::lcm_assign(), Parma_Polyhedra_-Library::Grid_Generator_System::num_rows(), and Parma_Polyhedra_Library::Grid_Generator_-System::space_dimension().

Referenced by add_grid_generator(), add_recycled_grid_generators(), add_space_dimensions(), add_-space_dimensions_and_project(), affine_image(), construct(), generalized_affine_image(), generalized_-affine_preimage(), normalize_divisors(), time_elapse_assign(), and upper_bound_assign().

11.71.5.95 bool Parma_Polyhedra_Library::Grid::OK (bool *check_not_empty* = `false`) const

Checks if all the invariants are satisfied.

Returns

`true` if and only if `*this` satisfies all the invariants and either `check_not_empty` is `false` or `*this` is not empty.

Parameters

check_not_empty `true` if and only if, in addition to checking the invariants, `*this` must be checked to be not empty.

The check is performed so as to intrude as little as possible. If the library has been compiled with run-time assertions enabled, error messages are written on `std::cerr` in case invariants are violated. This is useful for the purpose of debugging the library.

Definition at line 864 of file Grid_public.cc.

References ascii_dump(), Parma_Polyhedra_Library::Congruence_System::ascii_dump(), Parma_-Polyhedra_Library::Grid_Generator_System::ascii_dump(), clear_generators_up_to_date(), con_-sys, CON_VIRTUAL, congruences_are_minimized(), congruences_are_up_to_date(), dim_kinds, EQUALITY, gen_sys, GEN_VIRTUAL, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Grid_Generator_-System::has_points(), Parma_Polyhedra_Library::Congruence_System::is_equal_to(), Parma_-Polyhedra_Library::Grid_Generator::is_equal_to(), LINE, lower_triangular(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Grid_Generator_-System::num_rows(), Parma_Polyhedra_Library::Congruence_System::OK(), Parma_Polyhedra_-Library::Grid_Generator_System::OK(), Parma_Polyhedra_Library::Grid::Status::OK(), PARAMETER, PROPER_CONGRUENCE, simplify(), Parma_Polyhedra_Library::Grid_Generator::size(), space_-dim, Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), Parma_Polyhedra_-Library::Congruence_System::space_dimension(), status, swap(), update_generators(), and upper_-triangular().

Referenced by add_congruence_no_check(), add_grid_generator(), add_recycled_congruences(), add_-recycled_grid_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), bounded_affine_image(), bounded_affine_preimage(), concatenate_-assign(), congruence_widening_assign(), construct(), difference_assign(), drop_some_non_integer_-points(), expand_space_dimension(), fold_space_dimensions(), generalized_affine_image(), generalized_-affine_preimage(), generator_widening_assign(), Grid(), intersection_assign(), is_included_in(), limited_-congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_-assign(), map_space_dimensions(), minimize(), remove_higher_space_dimensions(), remove_space_-dimensions(), simplify_using_context_assign(), time_elapse_assign(), unconstrain(), and upper_bound_-assign().

11.71.5.96 PPL::Grid & Parma_Polyhedra_Library::Grid::operator= (const Grid & *y*)

The assignment operator. (`*this` and `y` can be dimension-incompatible.).

Definition at line 240 of file Grid_public.cc.

References con_sys, congruences_are_up_to_date(), dim_kinds, gen_sys, generators_are_up_to_date(), marked_empty(), set_empty(), set_zero_dim_univ(), space_dim, and status.

11.71.5.97 void Parma_Polyhedra_Library::Grid::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.71.5.98 PPL::Grid::Three_Valued_Boolean Parma_Polyhedra_Library::Grid::quick_equivalence_test (const Grid & *y*) const [private]

Polynomial but incomplete equivalence test between grids.

Definition at line 182 of file Grid_nonpublic.cc.

References con_sys, congruences_are_minimized(), gen_sys, generators_are_minimized(), marked_empty(), Parma_Polyhedra_Library::Congruence_System::num_equalities(), Parma_Polyhedra_Library::Grid_Generator_System::num_lines(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), space_dim, TVB_DONT_KNOW, TVB_FALSE, and TVB_TRUE.

Referenced by contains().

11.71.5.99 void Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality (Congruence & *row*, const Congruence & *pivot*, dimension_type *col*, Congruence_System & *sys*) [static, private]

Reduce `row` using `pivot`.

Use the equality `pivot` to change the representation of the congruence `row` such that element at index `col` of `row` is zero.

Definition at line 182 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::modulus(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by simplify().

11.71.5.100 void Parma_Polyhedra_Library::Grid::reduce_equality_with_equality (Congruence & *row*, const Congruence & *pivot*, dimension_type *col*) [static, private]

Reduces the equality `row` using the equality `pivot`.

Uses the equality `pivot` to change the representation of the equality `row` so that the element at index `col` of `row` is zero.

Definition at line 58 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by simplify().

11.71.5.101 void Parma_Polyhedra_Library::Grid::reduce_line_with_line (Grid_Generator & *row*, Grid_Generator & *pivot*, dimension_type *col*) [static, private]

Reduces the line `row` using the line `pivot`.

Uses the line `pivot` to change the representation of the line `row` so that the element at index `col` of `row` is zero.

Definition at line 32 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Grid_Generator::size(), and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by simplify().

11.71.5.102 void Parma_Polyhedra_Library::Grid::reduce_parameter_with_line (Grid_Generator & *row*, const Grid_Generator & *pivot*, dimension_type *col*, Grid_Generator_System & *sys*) [static, private]

Reduce `row` using `pivot`.

Use the line `pivot` to change the representation of the parameter `row` such that the element at index `col` of `row` is zero.

Definition at line 126 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Grid_Generator_System::num_columns(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by simplify().

11.71.5.103 template<typename R > void Parma_Polyhedra_Library::Grid::reduce_pc_with_pc (R & *row*, R & *pivot*, dimension_type *col*, dimension_type *start*, dimension_type *end*) [inline, static, private]

Reduces `row` using `pivot`.

Uses the point, parameter or proper congruence at `pivot` to change the representation of the point, parameter or proper congruence at `row` so that the element at index `col` of `row` is zero. Only elements from index `start` to index `end` are modified (i.e. it is assumed that all other elements are zero).

Definition at line 85 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcdext_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by simplify().

11.71.5.104 template<typename M , typename R > void Parma_Polyhedra_Library::Grid::reduce_reduced (M & *sys*, dimension_type *dim*, dimension_type *pivot_index*, dimension_type *start*, dimension_type *end*, const Dimension_Kinds & *dim_kinds*, bool *generators* = true) [inline, static, private]

Reduce column `dim` in rows preceding `pivot_index` in `sys`.

Required when converting (or simplifying) a congruence or generator system to "strong minimal form"; informally, strong minimal form means that, not only is the system in minimal form (ie a triangular matrix), but also the absolute values of the coefficients of the proper congruences and parameters are minimal. As a simple example, the set of congruences $\{3x \equiv_3 0, 4x + y \equiv_3 1\}$, (which is in minimal form) is equivalent to the set $\{3x \equiv_3 0, x + y \equiv_3 1\}$ (which is in strong minimal form).

Only consider from index `start` to index `end` of the row at `pivot_index`. Flag `generators` indicates whether `sys` is a congruence or generator system.

Definition at line 270 of file Grid.templates.hh.

References CON_VIRTUAL, EQUALITY, GEN_VIRTUAL, LINE, PARAMETER, PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::sub_mul_assign().

11.71.5.105 void Parma_Polyhedra_Library::Grid::refine_no_check (const Constraint & *c*) [private]

Uses the constraint `c` to refine `*this`.

Parameters

c The constraint to be added. Non-trivial inequalities are ignored.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 695 of file Grid_nonpublic.cc.

References add_congruence_no_check(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), marked_empty(), set_empty(), space_dim, and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by refine_with_constraint(), and refine_with_constraints().

11.71.5.106 void Parma_Polyhedra_Library::Grid::refine_with_congruence (const Congruence & *cg*) [inline]

Uses a copy of the congruence `cg` to refine `*this`.

Parameters

cg The congruence used.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 277 of file Grid.inlines.hh.

References add_congruence().

11.71.5.107 void Parma_Polyhedra_Library::Grid::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Uses a copy of the congruences in `cgs` to refine `*this`.

Parameters

cgs The congruences used.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 282 of file Grid.inlines.hh.

References add_congruences().

11.71.5.108 void Parma_Polyhedra_Library::Grid::refine_with_constraint (const Constraint & *c*)

Uses a copy of the constraint `c` to refine `*this`.

Parameters

c The constraint used. If it is not an equality, it will be ignored

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible.

Definition at line 1297 of file Grid_public.cc.

References marked_empty(), refine_no_check(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_dimension_incompatible().

11.71.5.109 void Parma_Polyhedra_Library::Grid::refine_with_constraints (const Constraint_System & *cs*)

Uses a copy of the constraints in `cs` to refine `*this`.

Parameters

cs The constraints used. Constraints that are not equalities are ignored.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 1307 of file Grid_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), marked_empty(), refine_no_check(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), and throw_dimension_incompatible().

11.71.5.110 PPL::Poly_Con_Relation Parma_Polyhedra_Library::Grid::relation_with (const Constraint & *c*) const

Returns the relations holding between `*this` and `c`.

Definition at line 578 of file Grid_public.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::begin(), Parma_Polyhedra_Library::Grid_Generator::coefficient(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Grid_Generator_System::end(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Grid_Generator::LINE, marked_empty(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Grid_Generator::PARAMETER, Parma_Polyhedra_Library::Grid_Generator::POINT, relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Grid_Generator::set_is_parameter(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Grid_Generator::size(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), throw_dimension_incompatible(), Parma_Polyhedra_Library::Constraint::type(), and update_generators().

11.71.5.111 PPL::Poly_Gen_Relation Parma_Polyhedra_Library::Grid::relation_with (const Generator & *g*) const

Returns the relations holding between `*this` and `g`.

Definition at line 538 of file Grid_public.cc.

References Parma_Polyhedra_Library::Generator::coefficient(), con_sys, congruences_are_up_to_date(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Generator::is_point(), marked_empty(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), space_dim, Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), throw_dimension_incompatible(), and update_congruences().

11.71.5.112 PPL::Poly_Gen_Relation Parma_Polyhedra_Library::Grid::relation_with (const Grid_Generator & *g*) const

Returns the relations holding between `*this` and `g`.

Definition at line 514 of file Grid_public.cc.

References con_sys, congruences_are_up_to_date(), marked_empty(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), space_dim, Parma_Polyhedra_Library::Grid_Generator::space_dimension(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), throw_dimension_incompatible(), and update_congruences().

11.71.5.113 PPL::Poly_Con_Relation Parma_Polyhedra_Library::Grid::relation_with (const Congruence & *cg*) const

Returns the relations holding between `*this` and `cg`.

Definition at line 361 of file Grid_public.cc.

References Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Grid_Generator_System::begin(), Parma_Polyhedra_Library::Grid_Generator_System::end(), Parma_Polyhedra_Library::gcd_assign(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Grid_Generator::LINE, marked_empty(), Parma_Polyhedra_Library::Congruence::modulus(), Parma_Polyhedra_Library::Grid_Generator::PARAMETER, Parma_Polyhedra_Library::Grid_Generator::POINT, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), throw_dimension_incompatible(), and update_generators().

Referenced by difference_assign(), limited_congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), relation_with(), and simplify_using_context_assign().

11.71.5.114 void Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions (dimension_type *new_dimension*)

Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`..

Exceptions

std::invalid_argument Thrown if `new_dimensions` is greater than the space dimension of `*this`.

Definition at line 306 of file Grid_chdims.cc.

References clear_congruences_up_to_date(), clear_generators_up_to_date(), con_sys, CON_VIRTUAL, congruences_are_minimized(), dim_kinds, Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Grid_Generator_System::erase_to_end(), gen_sys, GEN_VIRTUAL, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), is_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), OK(), Parma_Polyhedra_Library::Congruence_System::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Grid_Generator_System::remove_higher_space_dimensions(), set_empty(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Grid_Generator_System::swap(), Parma_Polyhedra_Library::swap(), Parma_Polyhedra_Library::Congruence_System::swap(), throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid_Generator_System::unset_pending_rows(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

Referenced by generalized_affine_image(), and generalized_affine_preimage().

11.71.5.115 void Parma_Polyhedra_Library::Grid::remove_space_dimensions (const Variables_Set & *vars*)

Removes all the specified dimensions from the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 262 of file Grid_chdims.cc.

References clear_congruences_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), marked_empty(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::remove_space_dimensions(), set_empty(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by fold_space_dimensions().

11.71.5.116 template<typename M , typename R > bool Parma_Polyhedra_Library::Grid::rows_are_zero (M & *system*, dimension_type *first*, dimension_type *last*, dimension_type *row_size*) [inline, static, private]

Checks that trailing rows contain only zero terms.

If all columns contain zero in the rows of `system` from row index `first` to row index `last` then return `true`, else return `false`. `row_size` gives the number of columns in each row.

This method is only used in assertions in the simplify methods.

Definition at line 238 of file Grid_simplify.cc.

11.71.5.117 void Parma_Polyhedra_Library::Grid::select_wider_congruences (const Grid & *y*, Congruence_System & *selected_cgs*) const [private]

Copies a widened selection of congruences from `y` to `selected_cgs`.

Definition at line 35 of file Grid_widenings.cc.

References con_sys, CON_VIRTUAL, congruences_are_minimized(), dim_kinds, EQUALITY, Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Congruence::is_equal_at_dimension(), marked_empty(), PROPER_CONGRUENCE, space_dim, and Parma_Polyhedra_Library::Congruence_System::space_dimension().

Referenced by congruence_widening_assign().

11.71.5.118 void Parma_Polyhedra_Library::Grid::select_wider_generators (const Grid & *y*, Grid_Generator_System & *widened_ggs*) const [private]

Copies widened generators from `y` to `widened_ggs`.

Definition at line 230 of file Grid_widenings.cc.

References Parma_Polyhedra_Library::Grid_Generator::coefficient(), dim_kinds, gen_sys, GEN_VIRTUAL, generators_are_minimized(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Grid_Generator::is_equal_at_dimension(), LINE, marked_empty(), PARAMETER, space_dim, Parma_Polyhedra_Library::Grid_Generator::space_dimension(), and Parma_Polyhedra_Library::Grid_Generator_System::space_dimension().

Referenced by generator_widening_assign().

11.71.5.119 void Parma_Polyhedra_Library::Grid::set_congruences_minimized () [inline, private]

Sets `status` to express that congruences are minimized.

Definition at line 70 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::set_c_minimized(), set_congruences_up_to_date(), and status.

Referenced by congruence_widening_assign(), construct(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), is_empty(), minimize(), minimized_congruences(), and update_generators().

11.71.5.120 void Parma_Polyhedra_Library::Grid::set_congruences_up_to_date () [inline, private]

Sets `status` to express that congruences are up-to-date.

Definition at line 65 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::set_c_up_to_date(), and status.

Referenced by add_congruence_no_check(), construct(), Grid(), and set_congruences_minimized().

11.71.5.121 void Parma_Polyhedra_Library::Grid::set_empty () [private]

Sets `status` to express that the grid is empty, clearing all corresponding matrices.

Definition at line 449 of file Grid_nonpublic.cc.

References con_sys, gen_sys, Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Grid::Status::set_empty(), space_dim, status, swap(), Parma_Polyhedra_Library::Grid_Generator_System::swap(), and Parma_Polyhedra_Library::Congruence::zero_dim_false().

Referenced by add_congruence_no_check(), add_constraint_no_check(), add_recycled_congruences(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), concatenate_assign(), congruence_widening_assign(), difference_assign(), Grid(), intersection_assign(), is_empty(), map_space_dimensions(), minimized_congruences(), operator=(), refine_no_check(), remove_higher_space_dimensions(), remove_space_dimensions(), time_elapse_assign(), update_generators(), and wrap_assign().

11.71.5.122 void Parma_Polyhedra_Library::Grid::set_generators_minimized () [inline, private]

Sets `status` to express that generators are minimized.

Definition at line 76 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::set_g_minimized(), set_generators_up_to_date(), and status.

Referenced by construct(), generator_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), max_min(), minimize(), minimized_grid_generators(), and update_generators().

11.71.5.123 void Parma_Polyhedra_Library::Grid::set_generators_up_to_date () [inline, private]

Sets `status` to express that generators are up-to-date.

Definition at line 60 of file Grid.inlines.hh.

References Parma_Polyhedra_Library::Grid::Status::set_g_up_to_date(), and status.

Referenced by add_grid_generator(), add_recycled_grid_generators(), construct(), Grid(), and set_-generators_minimized().

11.71.5.124 void Parma_Polyhedra_Library::Grid::set_zero_dim_univ () [private]

Sets `status` to express that the grid is the universe 0-dimension vector space, clearing all corresponding matrices.

Definition at line 440 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::clear(), Parma_Polyhedra_-Library::Congruence_System::clear(), con_sys, gen_sys, Parma_Polyhedra_Library::Grid_Generator_-System::insert(), Parma_Polyhedra_Library::Grid::Status::set_zero_dim_univ(), space_dim, and status.

Referenced by add_grid_generator(), add_recycled_grid_generators(), construct(), Grid(), map_-space_dimensions(), operator=(), remove_higher_space_dimensions(), remove_space_dimensions(), and simplify_using_context_assign().

11.71.5.125 void Parma_Polyhedra_Library::Grid::simplify (Grid_Generator_System & *gs*, Dimension_Kinds & *dim_kinds*) [static, private]

Converts `gs` to lower triangular (i.e. minimized) form.

Expects `gs` to contain at least one point.

Definition at line 251 of file Grid_simplify.cc.

References GEN_VIRTUAL, Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_-Library::Grid_Generator::is_line(), Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_-point(), LINE, Parma_Polyhedra_Library::Grid_Generator::negate(), Parma_Polyhedra_Library::Grid_-Generator_System::num_columns(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), Parma_Polyhedra_Library::Grid_Generator_System::OK(), PARAMETER, reduce_line_with_line(), reduce_parameter_with_line(), reduce_pc_with_pc(), swap(), and Parma_Polyhedra_Library::Grid_-Generator_System::unset_pending_rows().

11.71.5.126 bool Parma_Polyhedra_Library::Grid::simplify (Congruence_System & *cgs*, Dimension_Kinds & *dim_kinds*) [static, private]

Converts `cgs` to upper triangular (i.e. minimized) form.

Returns `true` if `cgs` represents the empty set, otherwise returns `false`.

Definition at line 376 of file Grid_simplify.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), CON_VIRTUAL, EQUALITY, Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_-Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Congruence::is_-equality(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_-Library::Congruence::modulus(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_-Polyhedra_Library::Congruence::negate(), Parma_Polyhedra_Library::Congruence_System::normalize_-moduli(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_-rows(), Parma_Polyhedra_Library::Congruence_System::OK(), PROPER_CONGRUENCE, Parma_-

Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, reduce_congruence_-with_equality(), reduce_equality_with_equality(), reduce_pc_with_pc(), Parma_Polyhedra_-Library::Matrix::rows, Parma_Polyhedra_Library::Congruence::set_is_equality(), and swap().

Referenced by congruence_widening_assign(), generator_widening_assign(), Parma_Polyhedra_-Library::Grid_Certificate::Grid_Certificate(), is_empty(), max_min(), minimize(), minimized_-congruences(), minimized_grid_generators(), OK(), and update_generators().

11.71.5.127 bool Parma_Polyhedra_Library::Grid::simplify_using_context_assign (const Grid & *y*)

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 1568 of file Grid_public.cc.

References add_congruence(), add_congruences(), add_recycled_congruences(), add_space_dimensions_-and_embed(), Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Grid_-Generator_System::begin(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), con_sys, congruences_are_minimized(), congruences_are_up_to_date(), Parma_Polyhedra_Library::Grid_-Generator::divisor(), Parma_Polyhedra_Library::Grid_Generator_System::end(), gen_sys, generators_-are_minimized(), Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_-Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Congruence_-System::insert(), is_empty(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_-Library::Poly_Con_Relation::is_included(), is_included_in(), Parma_Polyhedra_Library::Grid_-Generator::is_parameter(), Parma_Polyhedra_Library::Grid_Generator::is_point(), Parma_Polyhedra_-Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::Checked::le, minimize(), Parma_Polyhedra_Library::Congruence::modulus(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, relation_-with(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_-dimension(), swap(), throw_dimension_incompatible(), and Parma_Polyhedra_Library::UNIVERSE.

11.71.5.128 dimension_type Parma_Polyhedra_Library::Grid::space_dimension () const `[inline]`

Returns the dimension of the vector space enclosing `*this`.

Definition at line 220 of file Grid.inlines.hh.

References space_dim.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), Parma_-Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_-Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_-Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), concatenate_assign(), drop_some_non_integer_points(), expand_space_dimension(), Parma_Polyhedra_Library::Grid_-Certificate::Grid_Certificate(), hash_code(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and throw_dimension_incompatible().

11.71.5.129 bool Parma_Polyhedra_Library::Grid::strictly_contains (const Grid & *y*) const [inline]

Returns `true` if and only if `*this` strictly contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 361 of file Grid.inlines.hh.

References contains().

11.71.5.130 void Parma_Polyhedra_Library::Grid::swap (Grid & *y*) [inline]

Swaps `*this` with grid `y`. (`*this` and `y` can be dimension-incompatible.).

Definition at line 245 of file Grid.inlines.hh.

References con_sys, dim_kinds, gen_sys, space_dim, and status.

Referenced by add_recycled_grid_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), congruence_widening_assign(), construct(), generator_widening_assign(), map_space_dimensions(), OK(), set_empty(), simplify(), simplify_using_context_assign(), and swap().

11.71.5.131 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, dimension_type *required_space_dim*) const [protected]

Definition at line 810 of file Grid_nonpublic.cc.

References space_dimension().

11.71.5.132 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *var_name*, Variable *var*) const [protected]

Definition at line 798 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Variable::space_dimension(), and space_dimension().

11.71.5.133 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *gs_name*, const Grid_Generator_System & *gs*) const [protected]

Definition at line 791 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::space_dimension(), and throw_dimension_incompatible().

11.71.5.134 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *cs_name*, const Constraint_System & *cs*) const [protected]

Definition at line 784 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint_System::space_dimension(), and throw_dimension_incompatible().

11.71.5.135 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *cgs_name*, const Congruence_System & *cgs*) const [protected]

Definition at line 777 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Congruence_System::space_dimension(), and throw_dimension_incompatible().

11.71.5.136 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *g_name*, const Generator & *g*) const [protected]

Definition at line 770 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Generator::space_dimension(), and throw_dimension_incompatible().

11.71.5.137 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *g_name*, const Grid_Generator & *g*) const [protected]

Definition at line 763 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Grid_Generator::space_dimension(), and throw_dimension_incompatible().

11.71.5.138 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *c_name*, const Constraint & *c*) const [protected]

Definition at line 756 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_dimension_incompatible().

11.71.5.139 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *cg_name*, const Congruence & *cg*) const [protected]

Definition at line 749 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and throw_dimension_-incompatible().

11.71.5.140 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *e_name*, const Linear_Expression & *e*) const [protected]

Definition at line 742 of file Grid_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and throw_dimension_-incompatible().

11.71.5.141 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *gr_name*, const Grid & *gr*) const [protected]

Definition at line 735 of file Grid_nonpublic.cc.

References space_dimension(), and throw_dimension_incompatible().

11.71.5.142 void Parma_Polyhedra_Library::Grid::throw_dimension_incompatible (const char * *method*, const char * *other_name*, dimension_type *other_dim*) const [protected]

Definition at line 724 of file Grid_nonpublic.cc.

References space_dimension().

Referenced by add_congruence(), add_congruences(), add_constraint(), add_constraints(), add_grid_-generator(), add_recycled_congruences(), add_recycled_grid_generators(), affine_image(), affine_-preimage(), bounded_affine_image(), bounded_affine_preimage(), bounds(), congruence_widening_-assign(), constrains(), contains(), difference_assign(), drop_some_non_integer_points(), expand_space_-dimension(), fold_space_dimensions(), frequency(), generalized_affine_image(), generalized_affine_-preimage(), generator_widening_assign(), intersection_assign(), is_disjoint_from(), limited_congruence_-extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), refine_-with_constraint(), refine_with_constraints(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), simplify_using_context_assign(), throw_dimension_incompatible(), time_elapse_assign(), unconstrain(), upper_bound_assign(), upper_bound_assign_if_exact(), widening_-assign(), and wrap_assign().

11.71.5.143 void Parma_Polyhedra_Library::Grid::throw_invalid_argument (const char * *method*, const char * *reason*) const [protected]

Definition at line 715 of file Grid_nonpublic.cc.

Referenced by affine_image(), affine_preimage(), bounded_affine_image(), bounded_affine_preimage(), fold_space_dimensions(), generalized_affine_image(), generalized_affine_preimage(), and map_space_-dimensions().

11.71.5.144 void Parma_Polyhedra_Library::Grid::throw_invalid_constraint (const char * *method*, const char * *c_name*) const [protected]

Definition at line 829 of file Grid_nonpublic.cc.

Referenced by add_constraint_no_check(), and Grid().

11.71.5.145 void Parma_Polyhedra_Library::Grid::throw_invalid_constraints (const char * *method*, const char * *cs_name*) const [protected]

Definition at line 838 of file Grid_nonpublic.cc.

Referenced by Grid().

11.71.5.146 void Parma_Polyhedra_Library::Grid::throw_invalid_generator (const char * *method*, const char * *g_name*) const [protected]

Definition at line 848 of file Grid_nonpublic.cc.

Referenced by add_grid_generator().

11.71.5.147 void Parma_Polyhedra_Library::Grid::throw_invalid_generators (const char * *method*, const char * *gs_name*) const [protected]

Definition at line 858 of file Grid_nonpublic.cc.

Referenced by add_recycled_grid_generators(), and construct().

11.71.5.148 void Parma_Polyhedra_Library::Grid::throw_runtime_error (const char * *method*) const [protected]

Definition at line 708 of file Grid_nonpublic.cc.

11.71.5.149 void Parma_Polyhedra_Library::Grid::throw_space_dimension_overflow (const char * *method*, const char * *reason*) [static, protected]

Definition at line 820 of file Grid_nonpublic.cc.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), concatenate_assign(), expand_space_dimension(), and Grid().

11.71.5.150 void Parma_Polyhedra_Library::Grid::time_elapse_assign (const Grid & *y*)

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2435 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Grid_Generator::is_point(), marked_empty(), normalize_divisors(), Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), set_empty(), Parma_Polyhedra_Library::Grid_Generator::set_is_parameter(), space_dim, throw_dimension_incompatible(), and update_generators().

11.71.5.151 void Parma_Polyhedra_Library::Grid::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 367 of file Grid.inlines.hh.

11.71.5.152 memory_size_type Parma_Polyhedra_Library::Grid::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 225 of file Grid.inlines.hh.

References external_memory_in_bytes().

11.71.5.153 void Parma_Polyhedra_Library::Grid::unconstrain (const Variables_Set & *vars*)

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 1339 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), marked_empty(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), throw_dimension_incompatible(), and update_generators().

11.71.5.154 void Parma_Polyhedra_Library::Grid::unconstrain (Variable *var*)

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 1318 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), marked_empty(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by wrap_assign().

11.71.5.155 void Parma_Polyhedra_Library::Grid::update_congruences () const [private]

Updates and minimizes the congruences from the generators.

Definition at line 464 of file Grid_nonpublic.cc.

References gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Matrix::has_no_rows(), marked_empty(), space_dim, and Parma_Polyhedra_Library::Grid_Generator_System::space_dimension().

Referenced by add_congruence_no_check(), add_recycled_congruences(), concatenate_assign(), congruence_widening_assign(), congruences(), constrains(), intersection_assign(), is_included_in(), is_universe(), minimize(), and relation_with().

11.71.5.156 bool Parma_Polyhedra_Library::Grid::update_generators () const [private]

Updates and minimizes the generators from the congruences.

Returns

`false` if and only if `*this` turns out to be an empty grid.

It is illegal to call this method when the Status field already declares the grid to be empty.

Definition at line 490 of file Grid_nonpublic.cc.

References con_sys, congruences_are_minimized(), congruences_are_up_to_date(), conversion(), dim_kinds, gen_sys, marked_empty(), set_congruences_minimized(), set_empty(), set_generators_minimized(), simplify(), and space_dim.

Referenced by add_grid_generator(), add_recycled_grid_generators(), bounds(), Parma_Polyhedra_-Library::Box< ITV >::Box(), generalized_affine_image(), generalized_affine_preimage(), generator_-widening_assign(), grid_generators(), is_bounded(), is_discrete(), limited_congruence_extrapolation_-assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), map_space_-dimensions(), minimize(), minimized_grid_generators(), OK(), relation_with(), remove_space_-dimensions(), time_elapse_assign(), unconstrain(), and upper_bound_assign().

11.71.5.157 void Parma_Polyhedra_Library::Grid::upper_bound_assign (const Grid & *y*)

Assigns to `*this` the least upper bound of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1409 of file Grid_public.cc.

References clear_congruences_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_-up_to_date(), marked_empty(), normalize_divisors(), OK(), Parma_Polyhedra_Library::Grid_Generator_-System::recycling_insert(), space_dim, throw_dimension_incompatible(), and update_generators().

Referenced by difference_assign(), fold_space_dimensions(), and upper_bound_assign_if_exact().

11.71.5.158 bool Parma_Polyhedra_Library::Grid::upper_bound_assign_if_exact (const Grid & *y*)

If the upper bound of `*this` and `y` is exact it is assigned to `this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 1452 of file Grid_public.cc.

References difference_assign(), generators_are_up_to_date(), is_included_in(), marked_empty(), space_-dim, throw_dimension_incompatible(), and upper_bound_assign().

11.71.5.159 bool Parma_Polyhedra_Library::Grid::upper_triangular (const Grid_Generator_System & *sys*, const Dimension_Kinds & *dim_kinds*) [static, private]

If `sys` is upper triangular return `true`, else return `false`.

Definition at line 75 of file Grid_conversion.cc.

References GEN_VIRTUAL, Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), and Parma_Polyhedra_Library::Grid_Generator_System::space_dimension().

Referenced by conversion(), and OK().

11.71.5.160 void Parma_Polyhedra_Library::Grid::widening_assign (const Grid & *y*, unsigned * *tp* = NULL)

Assigns to `*this` the result of computing the Grid widening between `*this` and `y`.

This widening uses either the congruence or generator systems depending on which of the systems describing x and y are up to date and minimized.

Parameters

y A grid that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 432 of file Grid_widenings.cc.

References congruence_widening_assign(), congruences_are_up_to_date(), contains(), generator_widening_assign(), generators_are_up_to_date(), space_dim, and throw_dimension_incompatible().

Referenced by limited_congruence_extrapolation_assign(), and limited_extrapolation_assign().

11.71.5.161 void Parma_Polyhedra_Library::Grid::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true)

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system. This argument is for compatibility with wrap_assign() for the other domains and only checked for dimension-compatibility.

complexity_threshold A precision parameter of the wrapping operator. This argument is for compatibility with wrap_assign() for the other domains and is ignored.

wrap_individually `true` if the dimensions should be wrapped individually. As wrapping dimensions collectively does not improve the precision, this argument is ignored.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars` or with `*pcs`.

Warning

It is assumed that variables in `Vars` represent integers. Thus, where the extra cost is negligible, the integrality of these variables is enforced; possibly causing a non-integral grid to become empty.

Definition at line 2677 of file Grid_public.cc.

References add_congruence(), add_constraint(), add_grid_generator(), bounds_no_check(), Parma_Polyhedra_Library::Grid_Generator::coefficient(), Parma_Polyhedra_Library::Coefficient_one(), Parma_Polyhedra_Library::Grid_Generator::divisor(), frequency_no_check(), gen_sys, generators_are_minimized(), marked_empty(), minimize(), Parma_Polyhedra_Library::mul_2exp_assign(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::OVERFLOW_IMPOSSIBLE, Parma_Polyhedra_Library::OVERFLOW_UNDEFINED, Parma_Polyhedra_Library::OVERFLOW_WRAPS, PPL_DIRTY_TEMP_COEFFICIENT, set_empty(), Parma_Polyhedra_Library::SIGNED_2_COMPLEMENT, space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), throw_dimension_incompatible(), unconstrain(), and Parma_Polyhedra_Library::UNSIGNED.

11.71.6 Friends And Related Function Documentation

11.71.6.1 bool operator!= (const Grid & *x*, const Grid & *y*) [related]

Returns `true` if and only if `x` and `y` are different grids.

Note that `x` and `y` may be dimension-incompatible grids: in those cases, the value `true` is returned.

Definition at line 356 of file Grid.inlines.hh.

11.71.6.2 std::ostream & operator<< (std::ostream & *s*, const Grid & *gr*) [related]

Output operator.

Writes a textual representation of `gr` on `s`: `false` is written if `gr` is an empty grid; `true` is written if `gr` is a universe grid; a minimized system of congruences defining `gr` is written otherwise, all congruences in one row separated by ", "s.

Definition at line 2897 of file Grid_public.cc.

References is_empty(), is_universe(), and minimized_congruences().

11.71.6.3 bool operator== (const Grid & *x*, const Grid & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same grid.

Note that `x` and `y` may be dimension-incompatible grids: in those cases, the value `false` is returned.

Definition at line 2514 of file Grid_public.cc.

11.71.6.4 friend class Parma_Polyhedra_Library::Box [friend]

Definition at line 1924 of file Grid.defs.hh.

11.71.6.5 friend class Parma_Polyhedra_Library::Grid_Certificate [friend]

Definition at line 1922 of file Grid.defs.hh.

11.71.6.6 void swap (Parma_Polyhedra_Library::Grid & *x*, Parma_Polyhedra_Library::Grid & *y*) [related]

Specializes `std::swap`.

Definition at line 376 of file Grid.inlines.hh.

References swap().

11.71.7 Member Data Documentation

11.71.7.1 Congruence_System Parma_Polyhedra_Library::Grid::con_sys [private]

The system of congruences.

Definition at line 1966 of file Grid.defs.hh.

Referenced by add_congruence_no_check(), add_recycled_congruences(), add_space_dimensions_and_-embed(), add_space_dimensions_and_project(), affine_dimension(), affine_image(), affine_preimage(), ascii_dump(), concatenate_assign(), congruence_widening_assign(), congruences(), constrains(), construct(), external_memory_in_bytes(), Grid(), Parma_Polyhedra_Library::Grid_Certificate::Grid_-Certificate(), intersection_assign(), is_empty(), is_included_in(), is_universe(), map_space_dimensions(), minimize(), minimized_congruences(), OK(), operator=(), quick_equivalence_test(), relation_with(), remove_higher_space_dimensions(), select_wider_congruences(), set_empty(), set_zero_dim_univ(), simplify_using_context_assign(), swap(), and update_generators().

11.71.7.2 Dimension_Kinds Parma_Polyhedra_Library::Grid::dim_kinds [private]

Definition at line 1998 of file Grid.defs.hh.

Referenced by add_space_dimensions(), add_space_dimensions_and_embed(), add_space_dimensions_-and_project(), ascii_dump(), congruence_widening_assign(), construct(), generator_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), is_empty(), max_min(), minimize(), minimized_congruences(), minimized_grid_generators(), OK(), operator=(), remove_higher_space_-dimensions(), select_wider_congruences(), select_wider_generators(), swap(), and update_generators().

11.71.7.3 Grid_Generator_System Parma_Polyhedra_Library::Grid::gen_sys [private]

The system of generators.

Definition at line 1969 of file Grid.defs.hh.

Referenced by add_grid_generator(), add_recycled_grid_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_dimension(), affine_image(), affine_preimage(), ascii_dump(), bounds_no_check(), Parma_Polyhedra_Library::Box< ITV >::Box(), constrains(), construct(), external_memory_in_bytes(), frequency_no_check(), generalized_affine_image(), generalized_affine_preimage(), generator_widening_assign(), Grid(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), grid_generators(), is_bounded(), is_discrete(), map_space_dimensions(), max_min(), minimize(), minimized_grid_generators(), OK(), operator=(), quick_equivalence_test(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_wider_generators(), set_empty(), set_zero_dim_univ(), simplify_using_context_assign(), swap(), time_elapse_assign(), unconstrain(), update_congruences(), update_generators(), upper_bound_assign(), and wrap_assign().

11.71.7.4 dimension_type Parma_Polyhedra_Library::Grid::space_dim [private]

The number of dimensions of the enclosing vector space.

Definition at line 1979 of file Grid.defs.hh.

Referenced by add_congruence(), add_congruence_no_check(), add_congruences(), add_constraint(), add_constraint_no_check(), add_constraints(), add_grid_generator(), add_recycled_congruences(), add_recycled_grid_generators(), add_space_dimensions(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_dimension(), affine_image(), affine_preimage(), ascii_dump(), bounded_affine_image(), bounded_affine_preimage(), bounds(), bounds_no_check(), concatenate_assign(), congruence_widening_assign(), congruences(), constrains(), construct(), contains(), contains_integer_point(), difference_assign(), drop_some_non_integer_points(), expand_space_dimension(), fold_space_dimensions(), frequency(), frequency_no_check(), generalized_affine_image(), generalized_affine_preimage(), generator_widening_assign(), Grid(), grid_generators(), intersection_assign(), is_bounded(), is_discrete(), is_disjoint_from(), is_empty(), is_included_in(), is_universe(), limited_congruence_extrapolation_assign(), limited_extrapolation_assign(), limited_generator_extrapolation_assign(), map_space_dimensions(), max_min(), minimize(), minimized_grid_generators(), OK(), operator=(), quick_equivalence_test(), refine_no_check(), refine_with_constraint(), refine_with_constraints(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_wider_congruences(), select_wider_generators(), set_empty(), set_zero_dim_univ(), simplify_using_context_assign(), space_dimension(), swap(), time_elapse_assign(), unconstrain(), update_congruences(), update_generators(), upper_bound_assign(), upper_bound_assign_if_exact(), widening_assign(), and wrap_assign().

11.71.7.5 Status Parma_Polyhedra_Library::Grid::status [private]

The status flags to keep track of the grid's internal state.

Definition at line 1976 of file Grid.defs.hh.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), ascii_dump(), clear_congruences_minimized(), clear_congruences_up_to_date(), clear_empty(), clear_generators_minimized(), clear_generators_up_to_date(), congruences_are_minimized(), congruences_are_up_to_date(), construct(), generators_are_minimized(), generators_are_up_to_date(), Grid(), marked_empty(), OK(), operator=(), set_congruences_minimized(), set_congruences_up_to_date(), set_empty(), set_generators_minimized(), set_generators_up_to_date(), set_zero_dim_univ(), and swap().

The documentation for this class was generated from the following files:

- Grid.defs.hh
- Grid.inlines.hh
- Grid.templates.hh
- Grid_public.cc
- Grid_nonpublic.cc
- Grid_conversion.cc
- Grid_simplify.cc
- Grid_widenings.cc
- Grid_chdims.cc

11.72 Parma_Polyhedra_Library::Grid_Certificate Class Reference

The convergence certificate for the Grid widening operator.

`#include <Grid_Certificate.defs.hh>`

Classes

- struct Compare

  A total ordering on Grid certificates.

Public Member Functions

- Grid_Certificate ()

  Default constructor.

- Grid_Certificate (const Grid &gr)

  Constructor: computes the certificate for `gr`.

- Grid_Certificate (const Grid_Certificate &y)

  Copy constructor.

- ~Grid_Certificate ()

  Destructor.

- int compare (const Grid_Certificate &y) const

  The comparison function for certificates.

- int compare (const Grid &gr) const

  Compares `*this` *with the certificate for grid* `gr`.

- bool is_stabilizing (const Grid &gr) const

  Returns `true` *if and only if the certificate for grid* `gr` *is strictly smaller than* `*this`.

- bool OK () const

  Check if gathered information is meaningful.

Private Attributes

- dimension_type num_equalities
- dimension_type num_proper_congruences

11.72.1 Detailed Description

The convergence certificate for the Grid widening operator. Convergence certificates are used to instantiate the BHZ03 framework so as to define widening operators for the finite powerset domain.

Note

Each convergence certificate has to be used together with a compatible widening operator. In particular, Grid_Certificate can certify the Grid widening.

Definition at line 44 of file Grid_Certificate.defs.hh.

11.72.2 Constructor & Destructor Documentation

11.72.2.1 Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate () [inline]

Default constructor.

Definition at line 30 of file Grid_Certificate.inlines.hh.

References OK().

11.72.2.2 Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate (const Grid & *gr*)

Constructor: computes the certificate for `gr`.

Definition at line 35 of file Grid_Certificate.cc.

References Parma_Polyhedra_Library::Grid::con_sys, Parma_Polyhedra_Library::Grid::congruences_are_minimized(), Parma_Polyhedra_Library::Grid::congruences_are_up_to_date(), Parma_Polyhedra_Library::Grid::dim_kinds, Parma_Polyhedra_Library::Grid_Generator_System::empty(), Parma_Polyhedra_Library::Grid::gen_sys, Parma_Polyhedra_Library::Grid::generators_are_minimized(), Parma_Polyhedra_Library::Grid::generators_are_up_to_date(), Parma_Polyhedra_Library::Grid::marked_empty(), Parma_Polyhedra_Library::Congruence_System::num_equalities(), num_equalities, Parma_Polyhedra_Library::Grid_Generator_System::num_parameters(), Parma_Polyhedra_Library::Congruence_System::num_proper_congruences(), num_proper_congruences, Parma_Polyhedra_Library::Grid_Generator_System::num_rows(), Parma_Polyhedra_Library::Grid::set_congruences_minimized(), Parma_Polyhedra_Library::Grid::set_generators_minimized(), Parma_Polyhedra_Library::Grid::simplify(), and Parma_Polyhedra_Library::Grid::space_dimension().

11.72.2.3 Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate (const Grid_Certificate & *y*) [inline]

Copy constructor.

Definition at line 37 of file Grid_Certificate.inlines.hh.

11.72.2.4 Parma_Polyhedra_Library::Grid_Certificate::~Grid_Certificate () [inline]

Destructor.

Definition at line 43 of file Grid_Certificate.inlines.hh.

11.72.3 Member Function Documentation

11.72.3.1 int Parma_Polyhedra_Library::Grid_Certificate::compare (const Grid & *gr*) const

Compares `*this` with the certificate for grid `gr`.

Definition at line 101 of file Grid_Certificate.cc.

References compare().

11.72.3.2 int Parma_Polyhedra_Library::Grid_Certificate::compare (const Grid_Certificate & *y*) const

The comparison function for certificates.

Returns

-1, 0 or 1 depending on whether `*this` is smaller than, equal to, or greater than `y`, respectively.

Definition at line 89 of file Grid_Certificate.cc.

References num_equalities, num_proper_congruences, and OK().

Referenced by compare(), is_stabilizing(), and Parma_Polyhedra_Library::Grid_-Certificate::Compare::operator()().

11.72.3.3 bool Parma_Polyhedra_Library::Grid_Certificate::is_stabilizing (const Grid & *gr*) const [inline]

Returns `true` if and only if the certificate for grid `gr` is strictly smaller than `*this`.

Definition at line 47 of file Grid_Certificate.inlines.hh.

References compare().

11.72.3.4 bool Parma_Polyhedra_Library::Grid_Certificate::OK () const

Check if gathered information is meaningful.

Definition at line 107 of file Grid_Certificate.cc.

Referenced by compare(), and Grid_Certificate().

11.72.4 Member Data Documentation

11.72.4.1 dimension_type Parma_Polyhedra_Library::Grid_Certificate::num_equalities [private]

Number of a equalities in a minimized congruence system for the grid.

Definition at line 96 of file Grid_Certificate.defs.hh.

Referenced by compare(), and Grid_Certificate().

11.72.4.2 dimension_type Parma_Polyhedra_Library::Grid_Certificate::num_proper_congruences [private]

Number of a proper congruences in a minimized congruence system for the grid.

Definition at line 99 of file Grid_Certificate.defs.hh.

Referenced by compare(), and Grid_Certificate().

The documentation for this class was generated from the following files:

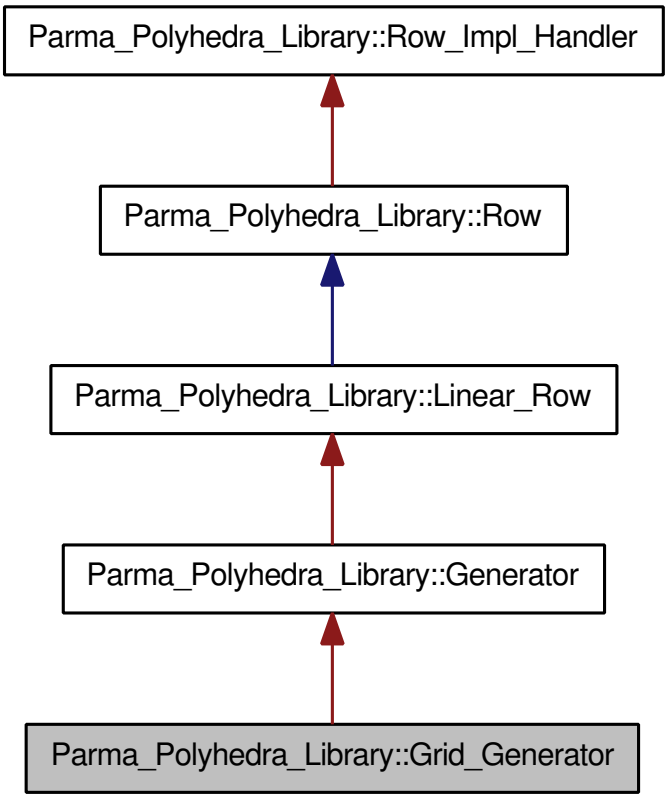
- Grid_Certificate.defs.hh
- Grid_Certificate.inlines.hh
- Grid_Certificate.cc

11.73 Parma_Polyhedra_Library::Grid_Generator Class Reference

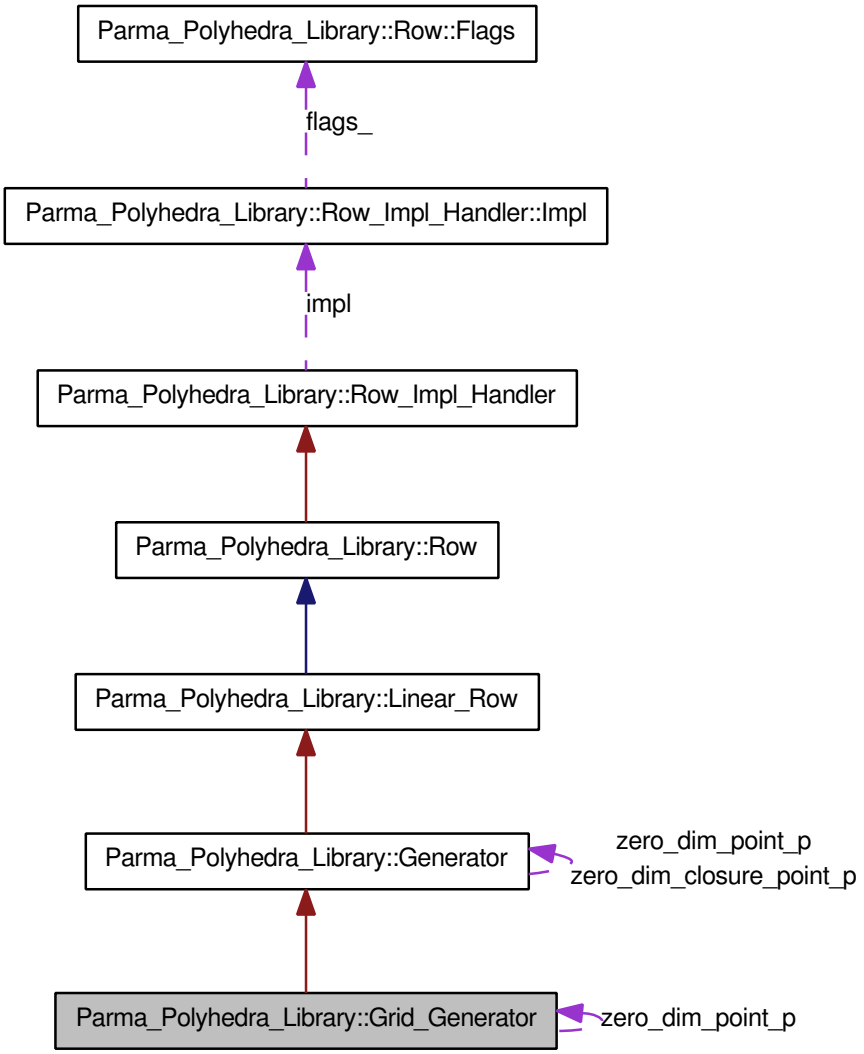
A grid line, parameter or grid point.

`#include <Grid_Generator.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Grid_Generator:



Collaboration diagram for Parma_Polyhedra_Library::Grid_Generator:



Public Types

- enum Type { LINE, PARAMETER, POINT }

  The generator type.

Public Member Functions

- Grid_Generator (const Grid_Generator &g)

  Ordinary copy constructor.

- ~Grid_Generator ()

  Destructor.

- Grid_Generator & operator= (const Grid_Generator &g)

Assignment operator.

- Grid_Generator & operator= (const Generator &g)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Type type () const

  Returns the generator type of `*this`.

- bool is_line () const

  Returns `true` *if and only if* `*this` *is a line.*

- bool is_parameter () const

  Returns `true` *if and only if* `*this` *is a parameter.*

- bool is_line_or_parameter () const

  Returns `true` *if and only if* `*this` *is a line or a parameter.*

- bool is_point () const

  Returns `true` *if and only if* `*this` *is a point.*

- bool is_parameter_or_point () const

  Returns `true` *if and only if* `*this` *row represents a parameter or a point.*

- Coefficient_traits::const_reference coefficient (Variable v) const

  Returns the coefficient of `v` *in* `*this`.

- Coefficient_traits::const_reference divisor () const

  Returns the divisor of `*this`.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool is_equivalent_to (const Grid_Generator &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are equivalent generators.*

- bool is_equal_to (const Grid_Generator &y) const

  Returns `true` *if* `*this` *is exactly equal to* `y`.

- bool is_equal_at_dimension (dimension_type dim, const Grid_Generator &gg) const

  Returns `true` *if* `*this` *is equal to* `gg` *in dimension* `dim`.

- bool all_homogeneous_terms_are_zero () const

  Returns `true` *if and only if all the homogeneous terms of* `*this` *are* 0.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void swap (Grid_Generator &y)

  Swaps `*this` *with* `y`.

- void coefficient_swap (Grid_Generator &y)

  Swaps `*this` *with* `y`*, leaving* `*this` *with the original capacity.*

Static Public Member Functions

- static Grid_Generator grid_line (const Linear_Expression &e)

  Returns the line of direction `e`.

- static Grid_Generator parameter (const Linear_Expression &e=Linear_Expression::zero(), Coefficient_traits::const_reference d=Coefficient_one())

  Returns the parameter of direction `e` *and size* `e/d`.

- static Grid_Generator grid_point (const Linear_Expression &e=Linear_Expression::zero(), Coefficient_traits::const_reference d=Coefficient_one())

  Returns the point at `e / d`.

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Grid_Generator can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Grid_Generator & zero_dim_point ()

  Returns the origin of the zero-dimensional space $\mathbb{R}^0$.

Private Member Functions

- void scale_to_divisor (Coefficient_traits::const_reference d)

  Scales `*this` *to be represented with a divisor of* `d` *(if* `\*this` *is a parameter or point).*

- Grid_Generator (Generator g)

  Constructs from polyhedron generator `g`*, stealing the underlying data structures from* `g`*.*

- dimension_type size () const

  Returns the actual size of `this`*.*

- void negate (dimension_type start, dimension_type end)

  Negates the elements from index `start` *to index* `end`*.*

- void set_divisor (Coefficient_traits::const_reference d)

  Sets the divisor of `*this` *to* `d`*.*

- void set_is_line ()

  Sets the Linear_Row kind to `LINE_OR_EQUALITY`*.*

- void set_is_parameter_or_point ()

  Sets the Linear_Row kind to `RAY_OR_POINT_OR_INEQUALITY`*.*

- void set_is_parameter ()

  Converts the Grid_Generator into a parameter.

- void strong_normalize ()

  Strong normalization: ensures that different Grid_Generator objects represent different hyperplanes or hyperspaces.

- Coefficient & operator[ ] (dimension_type k)

  Returns a reference to the element of the row indexed by `k`*.*

- Coefficient_traits::const_reference operator[ ] (dimension_type k) const

  Returns a constant reference to the element of the row indexed by `k`*.*

- void throw_invalid_argument (const char *method, const char *reason) const

  Throw a `std::invalid_argument` *exception containing the appropriate error message.*

Static Private Attributes

- static const Grid_Generator * zero_dim_point_p = 0

  Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$*.*

Friends

- class Grid
- class Grid_Generator_System
- class Grid_Generator_System::const_iterator
- class Congruence_System
- class Scalar_Products
- class Topology_Adjusted_Scalar_Product_Sign
- class Linear_Expression
- std::ostream & IO_Operators::operator<< (std::ostream &s, const Grid_Generator &g)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Grid_Generator &g)

  Output operator.

- void swap (Parma_Polyhedra_Library::Grid_Generator &x, Parma_Polyhedra_Library::Grid_Generator &y)

  Specializes `std::swap`.

- bool operator== (const Grid_Generator &x, const Grid_Generator &y)

  Returns `true` *if and only if* `x` *is equivalent to* `y`.

- bool operator!= (const Grid_Generator &x, const Grid_Generator &y)

  Returns `true` *if and only if* `x` *is not equivalent to* `y`.

- std::ostream & operator<< (std::ostream &s, const Grid_Generator::Type &t)

  Output operator.

11.73.1 Detailed Description

A grid line, parameter or grid point. An object of the class Grid_Generator is one of the following:

- a grid_line $\boldsymbol{l} = (a_0, \ldots, a_{n-1})^{\mathrm{T}}$;
- a parameter $\boldsymbol{q} = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$;
- a grid_point $\boldsymbol{p} = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$;

where n is the dimension of the space and, for grid_points and parameters, $d > 0$ is the divisor.

How to build a grid generator.

Each type of generator is built by applying the corresponding function (`grid_line`, `parameter` or `grid_point`) to a linear expression; the space dimension of the generator is defined as the space dimension of the corresponding linear expression. Linear expressions used to define a generator should be homogeneous (any constant term will be simply ignored). When defining grid points and parameters, an optional Coefficient argument can be used as a common *divisor* for all the coefficients occurring in the provided linear expression; the default value for this argument is 1.

In all the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds a grid line with direction $x - y - z$ and having space dimension 3:

```
Grid_Generator l = grid_line(x - y - z);
```

By definition, the origin of the space is not a line, so that the following code throws an exception:

```
Grid_Generator l = grid_line(0*x);
```

Example 2

The following code builds the parameter as the vector $\boldsymbol{p} = (1, -1, -1)^{\mathrm{T}} \in \mathbb{R}^3$ which has the same direction as the line in Example 1:

```
Grid_Generator q = parameter(x - y - z);
```

Note that, unlike lines, for parameters, the length as well as the direction of the vector represented by the code is significant. Thus `q` is *not* the same as the parameter `q1` defined by

```
Grid_Generator q1 = parameter(2x - 2y - 2z);
```

By definition, the origin of the space is not a parameter, so that the following code throws an exception:

```
Grid_Generator q = parameter(0*x);
```

Example 3

The following code builds the grid point $\boldsymbol{p} = (1, 0, 2)^{\mathrm{T}} \in \mathbb{R}^3$:

```
Grid_Generator p = grid_point(1*x + 0*y + 2*z);
```

The same effect can be obtained by using the following code:

```
Grid_Generator p = grid_point(x + 2*z);
```

Similarly, the origin $\mathbf{0} \in \mathbb{R}^3$ can be defined using either one of the following lines of code:

```
Grid_Generator origin3 = grid_point(0*x + 0*y + 0*z);
Grid_Generator origin3_alt = grid_point(0*z);
```

Note however that the following code would have defined a different point, namely $\mathbf{0} \in \mathbb{R}^2$:

```
Grid_Generator origin2 = grid_point(0*y);
```

The following two lines of code both define the only grid point having space dimension zero, namely $\mathbf{0} \in \mathbb{R}^0$. In the second case we exploit the fact that the first argument of the function `point` is optional.

```
Grid_Generator origin0 = Generator::zero_dim_point();
Grid_Generator origin0_alt = grid_point();
```

Example 4

The grid point $\boldsymbol{p}$ specified in Example 3 above can also be obtained with the following code, where we provide a non-default value for the second argument of the function `grid_point` (the divisor):

```
Grid_Generator p = grid_point(2*x + 0*y + 4*z, 2);
```

Obviously, the divisor can be used to specify points having some non-integer (but rational) coordinates. For instance, the grid point $\boldsymbol{p1} = (-1.5, 3.2, 2.1)^{\mathrm{T}} \in \mathbb{R}^3$ can be specified by the following code:

```
Grid_Generator p1 = grid_point(-15*x + 32*y + 21*z, 10);
```

If a zero divisor is provided, an exception is thrown.

Example 5

Parameters, like grid points can have a divisor. For instance, the parameter $\boldsymbol{q} = (1, 0, 2)^{\mathrm{T}} \in \mathbb{R}^3$ can be defined:

```
Grid_Generator q = parameter(2*x + 0*y + 4*z, 2);
```

Also, the divisor can be used to specify parameters having some non-integer (but rational) coordinates. For instance, the parameter $\boldsymbol{q} = (-1.5, 3.2, 2.1)^{\mathrm{T}} \in \mathbb{R}^3$ can be defined:

```
Grid_Generator q = parameter(-15*x + 32*y + 21*z, 10);
```

If a zero divisor is provided, an exception is thrown.

How to inspect a grid generator

Several methods are provided to examine a grid generator and extract all the encoded information: its space dimension, its type and the value of its integer coefficients and the value of the denominator.

Example 6

The following code shows how it is possible to access each single coefficient of a grid generator. If `g1` is a grid point having coordinates $(a_0, \ldots, a_{n-1})^{\mathrm{T}}$, we construct the parameter `g2` having coordinates $(a_0, 2a_1, \ldots, (i+1)a_i, \ldots, na_{n-1})^{\mathrm{T}}$.

```
if (g1.is_point()) {
  cout << "Grid point g1: " << g1 << endl;
  Linear_Expression e;
  for (dimension_type i = g1.space_dimension(); i-- > 0; )
    e += (i + 1) * g1.coefficient(Variable(i)) * Variable(i);
  Grid_Generator g2 = parameter(e, g1.divisor());
  cout << "Parameter g2: " << g2 << endl;
}
else
  cout << "Grid Generator g1 is not a grid point." << endl;
```

Therefore, for the grid point

```
Grid_Generator g1 = grid_point(2*x - y + 3*z, 2);
```

we would obtain the following output:

```
Grid point g1: p((2*A - B + 3*C)/2)
Parameter g2: parameter((2*A - 2*B + 9*C)/2)
```

When working with grid points and parameters, be careful not to confuse the notion of *coefficient* with the notion of *coordinate*: these are equivalent only when the divisor is 1.

Definition at line 228 of file Grid_Generator.defs.hh.

11.73.2 Member Enumeration Documentation

11.73.2.1 enum Parma_Polyhedra_Library::Grid_Generator::Type

The generator type.

Enumerator:

LINE The generator is a grid line.

PARAMETER The generator is a parameter.

POINT The generator is a grid point.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 283 of file Grid_Generator.defs.hh.

11.73.3 Constructor & Destructor Documentation

11.73.3.1 Parma_Polyhedra_Library::Grid_Generator::Grid_Generator (const Grid_Generator & *g*) [inline]

Ordinary copy constructor.

Definition at line 30 of file Grid_Generator.inlines.hh.

Referenced by initialize().

11.73.3.2 Parma_Polyhedra_Library::Grid_Generator::~Grid_Generator () [inline]

Destructor.

Definition at line 35 of file Grid_Generator.inlines.hh.

11.73.3.3 Parma_Polyhedra_Library::Grid_Generator::Grid_Generator (Generator *g*) [inline, explicit, private]

Constructs from polyhedron generator `g`, stealing the underlying data structures from `g`.

The last column in `g` becomes the parameter divisor column of the new Grid_Generator.

Definition at line 39 of file Grid_Generator.inlines.hh.

References swap().

11.73.4 Member Function Documentation

11.73.4.1 bool Parma_Polyhedra_Library::Grid_Generator::all_homogeneous_terms_are_zero () const

Returns `true` if and only if all the homogeneous terms of `*this` are 0.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 241 of file Grid_Generator.cc.

References size().

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::insert().

11.73.4.2 void Parma_Polyhedra_Library::Grid_Generator::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 133 of file Grid_Generator.cc.

References Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, size(), and type().

11.73.4.3 void Parma_Polyhedra_Library::Grid_Generator::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

11.73.4.4 bool Parma_Polyhedra_Library::Grid_Generator::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 156 of file Grid_Generator.cc.

References Parma_Polyhedra_Library::Row::shrink(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Row::swap().

11.73.4.5 Coefficient_traits::const_reference Parma_Polyhedra_Library::Grid_Generator::coefficient (Variable *v*) const [inline]

Returns the coefficient of `v` in `*this`.

Exceptions

std::invalid_argument Thrown if the index of `v` is greater than or equal to the space dimension of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 164 of file Grid_Generator.inlines.hh.

References space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::Generator::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Grid::constrains(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), Parma_Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Grid::select_wider_generators(), and Parma_Polyhedra_Library::Grid::wrap_assign().

11.73.4.6 void Parma_Polyhedra_Library::Grid_Generator::coefficient_swap (Grid_Generator & *y*)

Swaps `*this` with `y`, leaving `*this` with the original capacity.

All elements up to and including the last element of the smaller of `*this` and `y` are swapped. The parameter divisor element of `y` is swapped with the divisor element of `*this`.

Definition at line 114 of file Grid_Generator.cc.

References is_line(), set_is_line(), Parma_Polyhedra_Library::Generator::set_is_ray_or_point(), size(), Parma_Polyhedra_Library::swap(), and swap().

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert().

11.73.4.7 Coefficient_traits::const_reference Parma_Polyhedra_Library::Grid_Generator::divisor () const [inline]

Returns the divisor of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` is a line.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 120 of file Grid_Generator.inlines.hh.

References is_line(), is_line_or_parameter(), operator[ ](), size(), and throw_invalid_argument().

Referenced by Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Grid::Grid(), is_equal_at_dimension(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), Parma_Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::Grid::normalize_divisors(), OK(), operator<<(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), scale_to_divisor(), Parma_Polyhedra_Library::Grid::simplify_using_context_assign(), and Parma_Polyhedra_Library::Grid::wrap_assign().

11.73.4.8 memory_size_type Parma_Polyhedra_Library::Grid_Generator::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 176 of file Grid_Generator.inlines.hh.

11.73.4.9 void Parma_Polyhedra_Library::Grid_Generator::finalize () [static]

Finalizes the class.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 276 of file Grid_Generator.cc.

References zero_dim_point_p.

11.73.4.10 PPL::Grid_Generator Parma_Polyhedra_Library::Grid_Generator::grid_line (const Linear_Expression & *e*) [inline, static]

Returns the line of direction `e`.

Shorthand for Grid_Generator Grid_Generator::grid_line(const Linear_Expression& e).

Exceptions

std::invalid_argument Thrown if the homogeneous part of `e` represents the origin of the vector space.

Definition at line 210 of file Grid_Generator.inlines.hh.

11.73.4.11 PPL::Grid_Generator Parma_Polyhedra_Library::Grid_Generator::grid_point (const Linear_Expression & *e* = Linear_Expression::zero(), Coefficient_traits::const_reference *d* = Coefficient_one()) [inline, static]

Returns the point at `e` / `d`.

Shorthand for Grid_Generator Grid_Generator::grid_point(const Linear_Expression& e, Coefficient_traits::const_reference d).

Both `e` and `d` are optional arguments, with default values Linear_Expression::zero() and Coefficient_one(), respectively.

Exceptions

std::invalid_argument Thrown if `d` is zero.

Definition at line 223 of file Grid_Generator.inlines.hh.

Referenced by initialize().

11.73.4.12 void Parma_Polyhedra_Library::Grid_Generator::initialize () [static]

Initializes the class.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 269 of file Grid_Generator.cc.

References Grid_Generator(), grid_point(), and zero_dim_point_p.

11.73.4.13 bool Parma_Polyhedra_Library::Grid_Generator::is_equal_at_dimension (dimension_type *dim*, const Grid_Generator & *gg*) const [inline]

Returns `true` if `*this` is equal to `gg` in dimension `dim`.

Definition at line 130 of file Grid_Generator.inlines.hh.

References divisor(), and operator[ ]().

Referenced by Parma_Polyhedra_Library::Grid::select_wider_generators().

11.73.4.14 bool Parma_Polyhedra_Library::Grid_Generator::is_equal_to (const Grid_Generator & *y*) const

Returns `true` if `*this` is exactly equal to `y`.

Definition at line 230 of file Grid_Generator.cc.

References is_parameter(), size(), and type().

Referenced by Parma_Polyhedra_Library::Grid::OK().

11.73.4.15 bool Parma_Polyhedra_Library::Grid_Generator::is_equivalent_to (const Grid_Generator & *y*) const

Returns `true` if and only if `*this` and `y` are equivalent generators.

Generators having different space dimensions are not equivalent.

Definition at line 201 of file Grid_Generator.cc.

References LINE, POINT, space_dimension(), and type().

Referenced by operator==().

11.73.4.16 bool Parma_Polyhedra_Library::Grid_Generator::is_line () const `[inline]`

Returns `true` if and only if `*this` is a line.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 86 of file Grid_Generator.inlines.hh.

Referenced by coefficient_swap(), Parma_Polyhedra_Library::Grid::constrains(), divisor(), Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Grid::multiply_grid(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), set_divisor(), set_is_parameter(), and Parma_Polyhedra_Library::Grid::simplify().

11.73.4.17 bool Parma_Polyhedra_Library::Grid_Generator::is_line_or_parameter () const `[inline]`

Returns `true` if and only if `*this` is a line or a parameter.

Definition at line 96 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::Generator::is_line_or_ray().

Referenced by Parma_Polyhedra_Library::Grid::add_grid_generator(), Parma_Polyhedra_-Library::Grid::bounds_no_check(), divisor(), Parma_Polyhedra_Library::Grid::is_bounded(), Parma_-Polyhedra_Library::Grid::relation_with(), set_divisor(), and set_is_parameter().

11.73.4.18 bool Parma_Polyhedra_Library::Grid_Generator::is_parameter () const [inline]

Returns `true` if and only if `*this` is a parameter.

Definition at line 91 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::Generator::is_ray().

Referenced by Parma_Polyhedra_Library::Grid::add_grid_generator(), Parma_Polyhedra_-Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), is_equal_to(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.73.4.19 bool Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point () const [inline]

Returns `true` if and only if `*this` row represents a parameter or a point.

Definition at line 106 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality().

Referenced by Parma_Polyhedra_Library::Grid::add_grid_generator(), Parma_Polyhedra_-Library::Grid::multiply_grid(), Parma_Polyhedra_Library::Grid::normalize_divisors(), Parma_-Polyhedra_Library::Grid::reduce_parameter_with_line(), scale_to_divisor(), and Parma_Polyhedra_-Library::Grid::simplify().

11.73.4.20 bool Parma_Polyhedra_Library::Grid_Generator::is_point () const [inline]

Returns `true` if and only if `*this` is a point.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 101 of file Grid_Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_-Library::Grid::simplify_using_context_assign(), and Parma_Polyhedra_Library::Grid::time_elapse_-assign().

11.73.4.21 dimension_type Parma_Polyhedra_Library::Grid_Generator::max_space_dimension () [inline, static]

Returns the maximum space dimension a Grid_Generator can handle.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 50 of file Grid_Generator.inlines.hh.

11.73.4.22 void Parma_Polyhedra_Library::Grid_Generator::negate (dimension_type *start*, dimension_type *end*) [inline, private]

Negates the elements from index `start` to index `end`.

Definition at line 158 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::neg_assign().

Referenced by Parma_Polyhedra_Library::Grid::simplify().

11.73.4.23 bool Parma_Polyhedra_Library::Grid_Generator::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 369 of file Grid_Generator.cc.

References divisor(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), LINE, PARAMETER, POINT, size(), and type().

11.73.4.24 Grid_Generator & Parma_Polyhedra_Library::Grid_Generator::operator= (const Generator & *g*) [inline]

Assignment operator.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 152 of file Grid_Generator.inlines.hh.

References operator=().

11.73.4.25 Grid_Generator & Parma_Polyhedra_Library::Grid_Generator::operator= (const Grid_Generator & *g*) [inline]

Assignment operator.

Definition at line 146 of file Grid_Generator.inlines.hh.

Referenced by operator=().

11.73.4.26 Coefficient_traits::const_reference Parma_Polyhedra_Library::Grid_Generator::operator[ ] (dimension_type *k*) const [inline, private]

Returns a constant reference to the element of the row indexed by `k`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 60 of file Grid_Generator.inlines.hh.

References operator[ ]().

11.73.4.27 Coefficient & Parma_Polyhedra_Library::Grid_Generator::operator[ ] (dimension_type *k*) [inline, private]

Returns a reference to the element of the row indexed by `k`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 55 of file Grid_Generator.inlines.hh.

Referenced by divisor(), is_equal_at_dimension(), operator[ ](), set_divisor(), and set_is_parameter().

11.73.4.28 PPL::Grid_Generator Parma_Polyhedra_Library::Grid_Generator::parameter (const Linear_Expression & *e* = `Linear_Expression::zero()`, Coefficient_traits::const_reference *d* = `Coefficient_one()`) [inline, static]

Returns the parameter of direction `e` and size `e/d`.

Shorthand for Grid_Generator Grid_Generator::parameter(const Linear_Expression& e, Coefficient_traits::const_reference d).

Both `e` and `d` are optional arguments, with default values Linear_Expression::zero() and Coefficient_one(), respectively.

Exceptions

std::invalid_argument Thrown if `d` is zero.

Definition at line 216 of file Grid_Generator.inlines.hh.

11.73.4.29 void Parma_Polyhedra_Library::Grid_Generator::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Generator.

11.73.4.30 void Parma_Polyhedra_Library::Grid_Generator::scale_to_divisor (Coefficient_traits::const_reference *d*) [private]

Scales `*this` to be represented with a divisor of `d` (if *this is a parameter or point).

It is assumed that `d` is a multiple of the current divisor.

Exceptions

std::invalid_argument Thrown if `d` is zero.

Definition at line 250 of file Grid_Generator.cc.

References divisor(), Parma_Polyhedra_Library::exact_div_assign(), is_parameter_or_point(), PPL_-DIRTY_TEMP_COEFFICIENT, set_divisor(), and size().

Referenced by Parma_Polyhedra_Library::Grid::Grid().

11.73.4.31 void Parma_Polyhedra_Library::Grid_Generator::set_divisor (Coefficient_traits::const_reference *d*) [inline, private]

Sets the divisor of `*this` to `d`.

Exceptions

std::invalid_argument Thrown if `*this` is a line.

Definition at line 111 of file Grid_Generator.inlines.hh.

References is_line(), is_line_or_parameter(), operator[ ](), and size().

Referenced by scale_to_divisor().

11.73.4.32 void Parma_Polyhedra_Library::Grid_Generator::set_is_line () [inline, private]

Sets the Linear_Row kind to `LINE_OR_EQUALITY`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 136 of file Grid_Generator.inlines.hh.

Referenced by coefficient_swap(), and Parma_Polyhedra_Library::Grid::conversion().

11.73.4.33 void Parma_Polyhedra_Library::Grid_Generator::set_is_parameter () [private]

Converts the Grid_Generator into a parameter.

Definition at line 190 of file Grid_Generator.cc.

References is_line(), is_line_or_parameter(), operator[ ](), set_is_parameter_or_point(), and size().

Referenced by Parma_Polyhedra_Library::Grid::relation_with(), and Parma_Polyhedra_-Library::Grid::time_elapse_assign().

11.73.4.34 void Parma_Polyhedra_Library::Grid_Generator::set_is_parameter_or_point () [inline, private]

Sets the Linear_Row kind to `RAY_OR_POINT_OR_INEQUALITY`.

Definition at line 141 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::Generator::set_is_ray_or_point().

Referenced by Parma_Polyhedra_Library::Grid::conversion(), and set_is_parameter().

11.73.4.35 dimension_type Parma_Polyhedra_Library::Grid_Generator::size () const [inline, private]

Returns the actual size of `this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 45 of file Grid_Generator.inlines.hh.

Referenced by all_homogeneous_terms_are_zero(), ascii_dump(), Parma_Polyhedra_Library::Scalar_Products::assign(), coefficient_swap(), divisor(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), is_equal_to(), Parma_Polyhedra_Library::Grid::OK(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), Parma_Polyhedra_Library::Grid::reduce_line_with_line(), Parma_Polyhedra_Library::Scalar_Products::reduced_assign(), Parma_Polyhedra_Library::Grid::relation_with(), scale_to_divisor(), set_divisor(), and set_is_parameter().

11.73.4.36 dimension_type Parma_Polyhedra_Library::Grid_Generator::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 65 of file Grid_Generator.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::add_grid_generator(), coefficient(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), is_equivalent_to(), operator<<(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), Parma_Polyhedra_Library::Grid::select_wider_generators(), and Parma_Polyhedra_Library::Grid::throw_dimension_incompatible().

11.73.4.37 void Parma_Polyhedra_Library::Grid_Generator::strong_normalize () [inline, private]

Strong normalization: ensures that different Grid_Generator objects represent different hyperplanes or hyperspaces.

Applies both Linear_Row::normalize() and Linear_Row::sign_normalize().

This is simply a wrapper around the Generator::strong_normalize, which means applying it to a parameter may change the parameter.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 187 of file Grid_Generator.inlines.hh.

11.73.4.38 void Parma_Polyhedra_Library::Grid_Generator::swap (Grid_Generator & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 192 of file Grid_Generator.inlines.hh.

Referenced by coefficient_swap(), Grid_Generator(), and swap().

11.73.4.39 void Parma_Polyhedra_Library::Grid_Generator::throw_invalid_argument (const char * *method*, const char * *reason*) const [private]

Throw a `std::invalid_argument` exception containing the appropriate error message.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 33 of file Grid_Generator.cc.

Referenced by divisor().

11.73.4.40 memory_size_type Parma_Polyhedra_Library::Grid_Generator::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 171 of file Grid_Generator.inlines.hh.

11.73.4.41 Grid_Generator::Type Parma_Polyhedra_Library::Grid_Generator::type () const [inline]

Returns the generator type of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 70 of file Grid_Generator.inlines.hh.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, LINE, Parma_Polyhedra_Library::Generator::LINE, PARAMETER, POINT, Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, and Parma_Polyhedra_Library::Generator::type().

Referenced by ascii_dump(), is_equal_to(), is_equivalent_to(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), OK(), and operator<<().

11.73.4.42 const Grid_Generator & Parma_Polyhedra_Library::Grid_Generator::zero_dim_point () [inline, static]

Returns the origin of the zero-dimensional space $\mathbb{R}^0$.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 181 of file Grid_Generator.inlines.hh.

References zero_dim_point_p.

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::initialize().

11.73.5 Friends And Related Function Documentation

11.73.5.1 friend class Congruence_System [friend]

Definition at line 479 of file Grid_Generator.defs.hh.

11.73.5.2 friend class Grid [friend]

Definition at line 476 of file Grid_Generator.defs.hh.

11.73.5.3 friend class Grid_Generator_System [friend]

Definition at line 477 of file Grid_Generator.defs.hh.

11.73.5.4 friend class Grid_Generator_System::const_iterator [friend]

Definition at line 478 of file Grid_Generator.defs.hh.

11.73.5.5 std::ostream& IO_Operators::operator<< (std::ostream & *s*, const Grid_Generator & *g*) [friend]

11.73.5.6 friend class Linear_Expression [friend]

Definition at line 482 of file Grid_Generator.defs.hh.

11.73.5.7 bool operator!= (const Grid_Generator & *x*, const Grid_Generator & *y*) [related]

Returns `true` if and only if `x` is not equivalent to `y`.

Definition at line 204 of file Grid_Generator.inlines.hh.

11.73.5.8 std::ostream & operator<< (std::ostream & *s*, const Grid_Generator::Type & *t*) [related]

Output operator.

Definition at line 350 of file Grid_Generator.cc.

References LINE, PARAMETER, and Parma_Polyhedra_Library::Generator::POINT.

11.73.5.9 std::ostream & operator<< (std::ostream & *s*, const Grid_Generator & *g*) [related]

Output operator.

Definition at line 284 of file Grid_Generator.cc.

References divisor(), LINE, Parma_Polyhedra_Library::neg_assign(), PARAMETER, POINT, PPL_-DIRTY_TEMP_COEFFICIENT, space_dimension(), and type().

11.73.5.10 bool operator== (const Grid_Generator & *x*, const Grid_Generator & *y*) [related]

Returns `true` if and only if `x` is equivalent to `y`.

Definition at line 198 of file Grid_Generator.inlines.hh.

References is_equivalent_to().

11.73.5.11 friend class Scalar_Products [friend]

Definition at line 480 of file Grid_Generator.defs.hh.

11.73.5.12 void swap (Parma_Polyhedra_Library::Grid_Generator & *x*, Parma_Polyhedra_Library::Grid_Generator & *y*) [related]

Specializes `std::swap`.

Definition at line 234 of file Grid_Generator.inlines.hh.

References swap().

11.73.5.13 friend class Topology_Adjusted_Scalar_Product_Sign [friend]

Definition at line 481 of file Grid_Generator.defs.hh.

11.73.6 Member Data Documentation

11.73.6.1 const PPL::Grid_Generator * Parma_Polyhedra_Library::Grid_Generator::zero_dim_point_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the origin of the zero-dimensional space $\mathbb{R}^0$.

Reimplemented from Parma_Polyhedra_Library::Generator.

Definition at line 403 of file Grid_Generator.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_point().

The documentation for this class was generated from the following files:

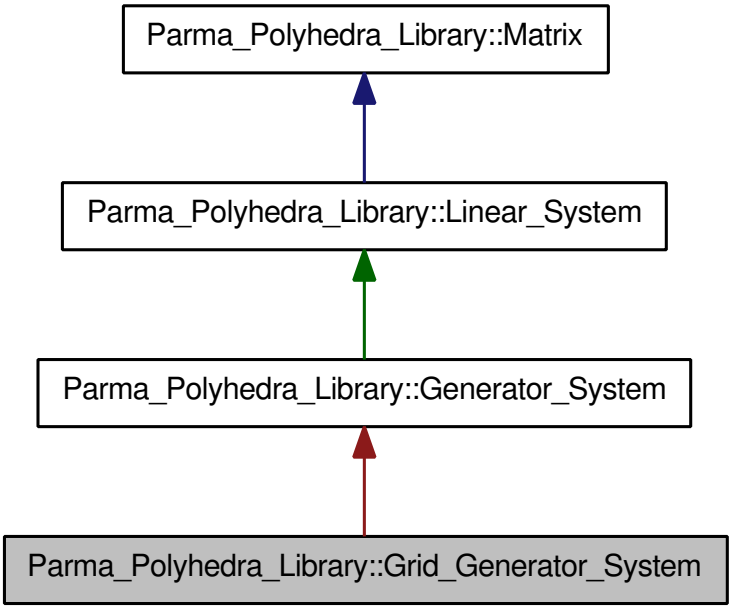
- Grid_Generator.defs.hh
- Grid_Generator.inlines.hh
- Grid_Generator.cc

11.74 Parma_Polyhedra_Library::Grid_Generator_System Class Reference

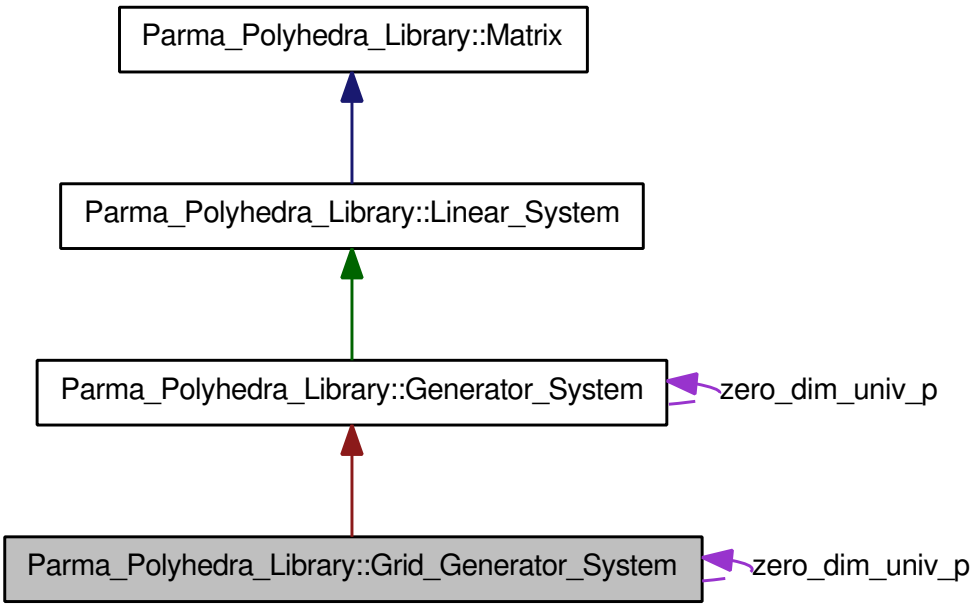
A system of grid generators.

`#include <Grid_Generator_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Grid_Generator_System:



Collaboration diagram for Parma_Polyhedra_Library::Grid_Generator_System:



Classes

- class const_iterator

  An iterator over a system of grid generators.

Public Member Functions

- Grid_Generator_System ()

  Default constructor: builds an empty system of generators.

- Grid_Generator_System (const Grid_Generator &g)

  Builds the singleton system containing only generator `g`*.*

- Grid_Generator_System (dimension_type dim)

  Builds an empty system of generators of dimension `dim`*.*

- Grid_Generator_System (const Grid_Generator_System &gs)

  Ordinary copy constructor.

- ~Grid_Generator_System ()

  Destructor.

- Grid_Generator_System & operator= (const Grid_Generator_System &y)

  Assignment operator.

- dimension_type space_dimension () const

Returns the dimension of the vector space enclosing `*this`.

- void clear ()

  Removes all the generators from the generator system and sets its space dimension to 0.

- void insert (const Grid_Generator &g)

  Inserts into `*this` *a copy of the generator* `g`*, increasing the number of space dimensions if needed.*

- void recycling_insert (Grid_Generator &g)

  Inserts into `*this` *the generator* `g`*, increasing the number of space dimensions if needed.*

- void recycling_insert (Grid_Generator_System &gs)

  Inserts into `*this` *the generators in* `gs`*, increasing the number of space dimensions if needed.*

- bool empty () const

  Returns `true` *if and only if* `*this` *has no generators.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first generator, if `this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- dimension_type num_rows () const

  Returns the number of rows (generators) in the system.

- dimension_type num_parameters () const

  Returns the number of parameters in the system.

- dimension_type num_lines () const

  Returns the number of lines in the system.

- bool has_points () const

  Returns `true` *if and only if* `*this` *contains one or more points.*

- bool is_equal_to (const Grid_Generator_System &y) const

  Returns `true` *if* `*this` *is identical to* `y`*.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`*.*

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`*.*

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`*.*

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void swap (Grid_Generator_System &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Grid_Generator_System can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Grid_Generator_System & zero_dim_univ ()

  Returns the singleton system containing only Grid_Generator::zero_dim_point().

Private Member Functions

- void set_sorted (bool b)

  Sets the sortedness flag of the system to `b`.

- void unset_pending_rows ()

  Sets the index to indicate that the system has no pending rows.

- void set_index_first_pending_row (dimension_type i)

  Sets the index of the first pending row to `i`.

- Grid_Generator & operator[ ] (dimension_type k)

  Returns the `k`*− th generator of the system.*

- const Grid_Generator & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k`*− th generator of the system.*

- void affine_image (dimension_type v, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator)

Assigns to a given variable an affine expression.

- void add_universe_rows_and_columns (dimension_type dims)

  Adds `dims` rows and `dims` columns of zeroes to the matrix, initializing the added rows as in the universe system.

- void remove_space_dimensions (const Variables_Set &vars)

  Removes all the specified dimensions from the generator system.

- void remove_higher_space_dimensions (dimension_type new_dimension)

  Removes the higher dimensions of the system so that the resulting system will have dimension `new_dimension`.

- void resize_no_copy (dimension_type new_num_rows, dimension_type new_num_columns)

  Resizes the system without worrying about the old contents.

- dimension_type num_columns () const

  Returns the number of columns of the matrix (i.e., the size of the rows).

- void erase_to_end (dimension_type first_to_erase)

  Erases from the matrix all the rows but those having an index less than `first_to_erase`.

- void permute_columns (const std::vector< dimension_type > &cycles)

  Permutes the columns of the matrix.

Static Private Attributes

- static const Grid_Generator_System * zero_dim_univ_p = 0

  Holds (between class initialization and finalization) a pointer to the singleton system containing only Grid_Generator::zero_dim_point().

Friends

- class Grid
- bool operator== (const Grid_Generator_System &x, const Grid_Generator_System &y)

  Returns `true` if and only if `x` and `y` are identical.

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Grid_Generator_System &gs)

  Output operator.

- void swap (Parma_Polyhedra_Library::Grid_Generator_System &x, Parma_Polyhedra_Library::Grid_Generator_System &y)

  Specializes `std::swap`.

11.74.1 Detailed Description

A system of grid generators. An object of the class Grid_Generator_System is a system of grid generators, i.e., a multiset of objects of the class Grid_Generator (lines, parameters and points). When inserting generators in a system, space dimensions are automatically adjusted so that all the generators in the system are defined on the same vector space. A system of grid generators which is meant to define a non-empty grid must include at least one point: the reason is that lines and parameters need a supporting point (lines only specify directions while parameters only specify direction and distance.

In all the examples it is assumed that variables `x` and `y` are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code defines the line having the same direction as the x axis (i.e., the first Cartesian axis) in $\mathbb{R}^2$:

```
Grid_Generator_System gs;
gs.insert(grid_line(x + 0*y));
```

As said above, this system of generators corresponds to an empty grid, because the line has no supporting point. To define a system of generators that does correspond to the x axis, we can add the following code which inserts the origin of the space as a point:

```
gs.insert(grid_point(0*x + 0*y));
```

Since space dimensions are automatically adjusted, the following code obtains the same effect:

```
gs.insert(grid_point(0*x));
```

In contrast, if we had added the following code, we would have defined a line parallel to the x axis through the point $(0, 1)^{\mathrm{T}} \in \mathbb{R}^2$.

```
gs.insert(grid_point(0*x + 1*y));
```

Example 2

The following code builds a system of generators corresponding to the grid consisting of all the integral points on the x axes; that is, all points satisfying the congruence relation

$$\{\, (x, 0)^{\mathrm{T}} \in \mathbb{R}^2 \mid x \pmod 1 \; 0 \,\},$$

```
Grid_Generator_System gs;
gs.insert(parameter(x + 0*y));
gs.insert(grid_point(0*x + 0*y));
```

Example 3

The following code builds a system of generators having three points corresponding to a non-relational grid consisting of all points whose coordinates are integer multiple of 3.

```
Grid_Generator_System gs;
gs.insert(grid_point(0*x + 0*y));
gs.insert(grid_point(0*x + 3*y));
gs.insert(grid_point(3*x + 0*y));
```

Example 4

By using parameters instead of two of the points we can define the same grid as that defined in the previous example. Note that there has to be at least one point and, for this purpose, any point in the grid could be considered. Thus the following code builds two identical grids from the grid generator systems `gs` and `gs1`.

```
Grid_Generator_System gs;
gs.insert(grid_point(0*x + 0*y));
gs.insert(parameter(0*x + 3*y));
gs.insert(parameter(3*x + 0*y));
Grid_Generator_System gs1;
gs1.insert(grid_point(3*x + 3*y));
gs1.insert(parameter(0*x + 3*y));
gs1.insert(parameter(3*x + 0*y));
```

Example 5

The following code builds a system of generators having one point and a parameter corresponding to all the integral points that lie on $x + y = 2$ in $\mathbb{R}^2$

```
Grid_Generator_System gs;
gs.insert(grid_point(1*x + 1*y));
gs.insert(parameter(1*x - 1*y));
```

Note

After inserting a multiset of generators in a grid generator system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* grid generator system will be available, where original generators may have been reordered, removed (if they are duplicate or redundant), etc.

Definition at line 179 of file Grid_Generator_System.defs.hh.

11.74.2 Constructor & Destructor Documentation

11.74.2.1 Parma_Polyhedra_Library::Grid_Generator_System::Grid_Generator_System () [inline]

Default constructor: builds an empty system of generators.

Definition at line 76 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, and set_sorted().

Referenced by initialize().

11.74.2.2 Parma_Polyhedra_Library::Grid_Generator_System::Grid_Generator_System (const Grid_Generator & *g*) [inline, explicit]

Builds the singleton system containing only generator `g`.

Definition at line 95 of file Grid_Generator_System.inlines.hh.

References set_sorted().

11.74.2.3 Parma_Polyhedra_Library::Grid_Generator_System::Grid_Generator_System (dimension_type *dim*) [inline, explicit]

Builds an empty system of generators of dimension `dim`.

Definition at line 88 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, and set_sorted().

11.74.2.4 Parma_Polyhedra_Library::Grid_Generator_System::Grid_Generator_System (const Grid_Generator_System & *gs*) [inline]

Ordinary copy constructor.

Definition at line 83 of file Grid_Generator_System.inlines.hh.

11.74.2.5 Parma_Polyhedra_Library::Grid_Generator_System::~Grid_Generator_System () [inline]

Destructor.

Definition at line 101 of file Grid_Generator_System.inlines.hh.

11.74.3 Member Function Documentation

11.74.3.1 void Parma_Polyhedra_Library::Grid_Generator_System::add_universe_rows_and_columns (dimension_type *dims*) [private]

Adds `dims` rows and `dims` columns of zeroes to the matrix, initializing the added rows as in the universe system.

Parameters

dims The number of rows and columns to be added: must be strictly positive.

Turns the $r \times c$ matrix A into the $(r + dims) \times (c + dims)$ matrix $\left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$ where B is the $dims \times dims$ unit matrix of the form $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 311 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Linear_Row::LINE_OR_EQUALITY, Parma_Polyhedra_Library::NECESSARILY_CLOSED, num_columns(), num_rows(), Parma_Polyhedra_Library::Matrix::swap_columns(), and unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Grid::add_space_dimensions(), and Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed().

11.74.3.2 void Parma_Polyhedra_Library::Grid_Generator_System::affine_image (dimension_type *v*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator*) [private]

Assigns to a given variable an affine expression.

Parameters

v Index of the column to which the affine transformation is assigned;

expr The numerator of the affine transformation: $\sum_{i=0}^{n-1} a_i x_i + b$;

denominator The denominator of the affine transformation;

We allow affine transformations (see the Section Operations on Rational Grids)to have rational coefficients. Since the coefficients of linear expressions are integers we also provide an integer `denominator` that will be used as denominator of the affine transformation. The denominator is required to be a positive integer and its default value is 1.

The affine transformation assigns to each element of `v` -th column the follow expression:

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{\text{denominator}}.$$

`expr` is a constant parameter and unaltered by this computation.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 163 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Scalar_Products::assign(), num_columns(), num_rows(), PPL_-DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), space_dimension(), and swap().

Referenced by Parma_Polyhedra_Library::Grid::affine_image(), and Parma_Polyhedra_-Library::Grid::affine_preimage().

11.74.3.3 void Parma_Polyhedra_Library::Grid_Generator_System::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 208 of file Grid_Generator_System.cc.

11.74.3.4 void Parma_Polyhedra_Library::Grid_Generator_System::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Referenced by Parma_Polyhedra_Library::Grid::ascii_dump(), and Parma_Polyhedra_-Library::Grid::OK().

11.74.3.5 bool Parma_Polyhedra_Library::Grid_Generator_System::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Resizes the matrix of generators using the numbers of rows and columns read from `s`, then initializes the coordinates of each generator and its type reading the contents from `s`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 216 of file Grid_Generator_System.cc.

References num_columns(), num_rows(), OK(), resize_no_copy(), set_index_first_pending_row(), and set_sorted().

11.74.3.6 Grid_Generator_System::const_iterator Parma_Polyhedra_Library::Grid_Generator_System::begin () const [inline]

Returns the const_iterator pointing to the first generator, if `this` is not empty; otherwise, returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 238 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::map_space_dimensions(), operator<<(), Parma_Polyhedra_Library::Grid::relation_with(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.74.3.7 void Parma_Polyhedra_Library::Grid_Generator_System::clear () [inline]

Removes all the generators from the generator system and sets its space dimension to 0.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 130 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), set_sorted(), and unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Grid::set_zero_dim_univ().

11.74.3.8 bool Parma_Polyhedra_Library::Grid_Generator_System::empty () const [inline]

Returns `true` if and only if `*this` has no generators.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 227 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate().

11.74.3.9 Grid_Generator_System::const_iterator Parma_Polyhedra_Library::Grid_Generator_System::end () const [inline]

Returns the past-the-end const_iterator.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 244 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::map_space_dimensions(), operator<<(), Parma_Polyhedra_Library::Grid::relation_with(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.74.3.10 void Parma_Polyhedra_Library::Grid_Generator_System::erase_to_end (dimension_type *first_to_erase*) [inline, private]

Erases from the matrix all the rows but those having an index less than `first_to_erase`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 59 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions().

11.74.3.11 memory_size_type Parma_Polyhedra_Library::Grid_Generator_System::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 144 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::external_memory_in_bytes().

11.74.3.12 void Parma_Polyhedra_Library::Grid_Generator_System::finalize () [static]

Finalizes the class.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 253 of file Grid_Generator_System.cc.

References zero_dim_univ_p.

11.74.3.13 bool Parma_Polyhedra_Library::Grid_Generator_System::has_points () const [inline]

Returns `true` if and only if `*this` contains one or more points.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 250 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::add_recycled_grid_generators(), Parma_Polyhedra_Library::Grid::construct(), and Parma_Polyhedra_Library::Grid::OK().

11.74.3.14 void Parma_Polyhedra_Library::Grid_Generator_System::initialize () [static]

Initializes the class.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 246 of file Grid_Generator_System.cc.

References Grid_Generator_System(), Parma_Polyhedra_Library::Grid_Generator::zero_dim_point(), and zero_dim_univ_p.

11.74.3.15 void Parma_Polyhedra_Library::Grid_Generator_System::insert (const Grid_Generator & *g*)

Inserts into `*this` a copy of the generator `g`, increasing the number of space dimensions if needed.

If `g` is an all-zero parameter then the only action is to ensure that the space dimension of `*this` is at least the space dimension of `g`.

Definition at line 85 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::add_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Grid_Generator::all_homogeneous_terms_are_zero(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Grid_Generator::is_parameter(), num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), num_rows(), OK(), Parma_Polyhedra_Library::Matrix::row_capacity, set_index_first_pending_row(), set_sorted(), Parma_Polyhedra_Library::Grid_Generator::size(), space_dimension(), Parma_Polyhedra_Library::Grid_Generator::space_dimension(), swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by Parma_Polyhedra_Library::Grid::add_grid_generator(), Parma_Polyhedra_Library::Grid::add_recycled_grid_generators(), Parma_Polyhedra_Library::Grid::add_space_dimensions(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Polyhedron::grid_generators(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), Parma_Polyhedra_Library::Grid::select_wider_generators(), and Parma_Polyhedra_Library::Grid::set_zero_dim_univ().

11.74.3.16 bool Parma_Polyhedra_Library::Grid_Generator_System::is_equal_to (const Grid_Generator_System & *y*) const [inline]

Returns `true` if `*this` is identical to `y`.

Definition at line 70 of file Grid_Generator_System.inlines.hh.

References operator==.

11.74.3.17 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::max_space_dimension () [inline, static]

Returns the maximum space dimension a Grid_Generator_System can handle.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 111 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::max_space_dimension().

11.74.3.18 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::num_columns () const [inline, private]

Returns the number of columns of the matrix (i.e., the size of the rows).

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 54 of file Grid_Generator_System.inlines.hh.

Referenced by add_universe_rows_and_columns(), affine_image(), ascii_load(), insert(), recycling_insert(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), remove_space_dimensions(), and Parma_Polyhedra_Library::Grid::simplify().

11.74.3.19 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::num_lines () const [inline]

Returns the number of lines in the system.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 164 of file Grid_Generator_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid::generator_widening_assign(), and Parma_Polyhedra_Library::Grid::quick_equivalence_test().

11.74.3.20 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::num_parameters () const [inline]

Returns the number of parameters in the system.

Definition at line 159 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Generator_System::num_rays().

Referenced by Parma_Polyhedra_Library::Grid::generator_widening_assign(), and Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate().

11.74.3.21 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::num_rows () const [inline]

Returns the number of rows (generators) in the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 154 of file Grid_Generator_System.inlines.hh.

Referenced by add_universe_rows_and_columns(), Parma_Polyhedra_Library::Grid::affine_dimension(), affine_image(), ascii_load(), Parma_Polyhedra_Library::Grid::bounds_no_check(), Parma_Polyhedra_Library::Grid::constrains(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Grid::generator_widening_assign(), Parma_Polyhedra_Library::Grid_Certificate::Grid_Certificate(), Parma_Polyhedra_Library::Grid::grid_generators(), insert(), Parma_Polyhedra_Library::Grid::is_bounded(), Parma_Polyhedra_Library::Grid::is_discrete(), Parma_Polyhedra_Library::Grid::minimized_grid_generators(), Parma_Polyhedra_Library::Grid::normalize_divisors(), Parma_Polyhedra_Library::Grid::OK(), OK(), Parma_Polyhedra_Library::Grid::quick_equivalence_test(), recycling_insert(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Grid::simplify(), Parma_Polyhedra_Library::Grid::time_elapse_assign(), and Parma_Polyhedra_Library::Grid::upper_triangular().

11.74.3.22 bool Parma_Polyhedra_Library::Grid_Generator_System::OK () const

Checks if all the invariants are satisfied.

Returns `true` if and only if `*this` is a valid Linear_System and each row in the system is a valid Grid_Generator.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 260 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, num_rows(), Parma_Polyhedra_Library::Matrix::OK(), and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by ascii_load(), insert(), Parma_Polyhedra_Library::Grid::OK(), remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid::simplify().

11.74.3.23 Grid_Generator_System & Parma_Polyhedra_Library::Grid_Generator_System::operator= (const Grid_Generator_System & *y*) [inline]

Assignment operator.

Definition at line 105 of file Grid_Generator_System.inlines.hh.

11.74.3.24 const Grid_Generator & Parma_Polyhedra_Library::Grid_Generator_System::operator[ ] (dimension_type *k*) const [inline, private]

Returns a constant reference to the k− th generator of the system.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 260 of file Grid_Generator_System.inlines.hh.

References operator[ ]().

11.74.3.25 Grid_Generator & Parma_Polyhedra_Library::Grid_Generator_System::operator[ ] (dimension_type *k*) [inline, private]

Returns the k− th generator of the system.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 255 of file Grid_Generator_System.inlines.hh.

Referenced by operator[ ](), and recycling_insert().

11.74.3.26 void Parma_Polyhedra_Library::Grid_Generator_System::permute_columns (const std::vector< dimension_type > & *cycles*) [inline, private]

Permutes the columns of the matrix.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 65 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Linear_System::permute_columns().

Referenced by Parma_Polyhedra_Library::Grid::map_space_dimensions().

11.74.3.27 void Parma_Polyhedra_Library::Grid_Generator_System::print () const

Prints *this to std::cerr using operator<<.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

11.74.3.28 void Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert (Grid_Generator_System & *gs*)

Inserts into *this the generators in gs, increasing the number of space dimensions if needed.

Definition at line 35 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_-Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, num_columns(), num_rows(), Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_-INEQUALITY, set_index_first_pending_row(), and Parma_Polyhedra_Library::Matrix::swap_columns().

11.74.3.29 void Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert (Grid_Generator & *g*)

Inserts into `*this` the generator `g`, increasing the number of space dimensions if needed.

Definition at line 61 of file Grid_Generator_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_-Library::Matrix::add_zero_rows_and_columns(), Parma_Polyhedra_Library::Grid_-Generator::coefficient_swap(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, num_-columns(), num_rows(), operator[ ](), Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_-OR_INEQUALITY, set_index_first_pending_row(), Parma_Polyhedra_Library::Grid_Generator::size(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::add_recycled_grid_generators(), Parma_-Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_-affine_preimage(), Parma_Polyhedra_Library::Grid::time_elapse_assign(), Parma_Polyhedra_-Library::Grid::unconstrain(), and Parma_Polyhedra_Library::Grid::upper_bound_assign().

11.74.3.30 void Parma_Polyhedra_Library::Grid_Generator_System::remove_-higher_space_dimensions (dimension_type *new_dimension*) [private]

Removes the higher dimensions of the system so that the resulting system will have dimension `new_-dimension`.

The value of `new_dimension` must be at most the space dimension of `*this`.

Definition at line 362 of file Grid_Generator_System.cc.

References OK(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), space_-dimension(), and Parma_Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions().

11.74.3.31 void Parma_Polyhedra_Library::Grid_Generator_System::remove_space_dimensions (const Variables_Set & *vars*) [private]

Removes all the specified dimensions from the generator system.

The space dimension of the variable with the highest space dimension in `vars` must be at most the space dimension of `this`.

Definition at line 328 of file Grid_Generator_System.cc.

References num_columns(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), space_dimension(), and Parma_-

Polyhedra_Library::Matrix::swap_columns().

Referenced by Parma_Polyhedra_Library::Grid::remove_space_dimensions().

11.74.3.32 void Parma_Polyhedra_Library::Grid_Generator_System::resize_no_copy (dimension_type *new_num_rows*, dimension_type *new_num_columns*) [inline, private]

Resizes the system without worrying about the old contents.

Parameters

new_num_rows The number of rows of the resized system;

new_num_columns The number of columns of the resized system.

The system is expanded to the specified dimensions avoiding reallocation whenever possible. The contents of the original system is lost.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 48 of file Grid_Generator_System.inlines.hh.

Referenced by ascii_load(), and Parma_Polyhedra_Library::Grid::conversion().

11.74.3.33 void Parma_Polyhedra_Library::Grid_Generator_System::set_index_first_pending_row (dimension_type *i*) [inline, private]

Sets the index of the first pending row to `i`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 43 of file Grid_Generator_System.inlines.hh.

Referenced by ascii_load(), Parma_Polyhedra_Library::Grid::conversion(), insert(), and recycling_insert().

11.74.3.34 void Parma_Polyhedra_Library::Grid_Generator_System::set_sorted (bool *b*) [inline, private]

Sets the sortedness flag of the system to `b`.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 33 of file Grid_Generator_System.inlines.hh.

Referenced by ascii_load(), clear(), Parma_Polyhedra_Library::Grid::Grid(), Grid_Generator_System(), insert(), and Parma_Polyhedra_Library::Grid::map_space_dimensions().

11.74.3.35 dimension_type Parma_Polyhedra_Library::Grid_Generator_System::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 117 of file Grid_Generator_System.inlines.hh.

References Parma_Polyhedra_Library::Generator_System::space_dimension().

Referenced by Parma_Polyhedra_Library::Grid::add_recycled_grid_generators(), Parma_Polyhedra_Library::Grid::add_space_dimensions(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project(), affine_image(), Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Grid::grid_generators(), insert(), Parma_Polyhedra_Library::Grid::minimized_grid_generators(), Parma_Polyhedra_Library::Grid::normalize_divisors(), Parma_Polyhedra_Library::Grid::OK(), remove_higher_space_dimensions(), remove_space_dimensions(), Parma_Polyhedra_Library::Grid::select_wider_generators(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::update_congruences(), and Parma_Polyhedra_Library::Grid::upper_triangular().

11.74.3.36 void Parma_Polyhedra_Library::Grid_Generator_System::swap (Grid_Generator_System & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 139 of file Grid_Generator_System.inlines.hh.

Referenced by affine_image(), insert(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Grid::set_empty(), and swap().

11.74.3.37 memory_size_type Parma_Polyhedra_Library::Grid_Generator_System::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 149 of file Grid_Generator_System.inlines.hh.

11.74.3.38 void Parma_Polyhedra_Library::Grid_Generator_System::unset_pending_rows () [inline, private]

Sets the index to indicate that the system has no pending rows.

Reimplemented from Parma_Polyhedra_Library::Linear_System.

Definition at line 38 of file Grid_Generator_System.inlines.hh.

Referenced by add_universe_rows_and_columns(), clear(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid::simplify().

11.74.3.39 const Grid_Generator_System & Parma_Polyhedra_Library::Grid_Generator_System::zero_dim_univ () [inline, static]

Returns the singleton system containing only Grid_Generator::zero_dim_point().

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 124 of file Grid_Generator_System.inlines.hh.

References zero_dim_univ_p.

11.74.4 Friends And Related Function Documentation

11.74.4.1 friend class Grid [friend]

Definition at line 385 of file Grid_Generator_System.defs.hh.

11.74.4.2 std::ostream & operator<< (std::ostream & *s*, const Grid_Generator_System & *gs*) [related]

Output operator.

Writes `false` if `gs` is empty. Otherwise, writes on `s` the generators of `gs`, all in one row and separated by ", ".

Definition at line 295 of file Grid_Generator_System.cc.

References begin(), and end().

11.74.4.3 bool operator== (const Grid_Generator_System & *x*, const Grid_Generator_System & *y*) [friend]

Returns `true` if and only if `x` and `y` are identical.

Definition at line 266 of file Grid_Generator_System.inlines.hh.

Referenced by is_equal_to(), and Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator==().

11.74.4.4 void swap (Parma_Polyhedra_Library::Grid_Generator_System & *x*, Parma_Polyhedra_Library::Grid_Generator_System & *y*) [related]

Specializes `std::swap`.

Definition at line 278 of file Grid_Generator_System.inlines.hh.

References swap().

11.74.5 Member Data Documentation

11.74.5.1 const PPL::Grid_Generator_System * Parma_Polyhedra_Library::Grid_Generator_System::zero_dim_univ_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the singleton system containing only Grid_Generator::zero_dim_point().

Reimplemented from Parma_Polyhedra_Library::Generator_System.

Definition at line 383 of file Grid_Generator_System.defs.hh.

Referenced by finalize(), initialize(), and zero_dim_univ().

The documentation for this class was generated from the following files:

- Grid_Generator_System.defs.hh
- Grid_Generator_System.inlines.hh
- Grid_Generator_System.cc

11.75 Parma_Polyhedra_Library::H79_Certificate Class Reference

A convergence certificate for the H79 widening operator.

`#include <H79_Certificate.defs.hh>`

Classes

- struct Compare

  A total ordering on H79 certificates.

Public Member Functions

- H79_Certificate ()

  Default constructor.

- template<typename PH >
  H79_Certificate (const PH &ph)

  Constructor: computes the certificate for `ph`.

- H79_Certificate (const Polyhedron &ph)

  Constructor: computes the certificate for `ph`.

- H79_Certificate (const H79_Certificate &y)

  Copy constructor.

- ~H79_Certificate ()

  Destructor.

- int compare (const H79_Certificate &y) const

The comparison function for certificates.

- template<typename PH >
  int compare (const PH &ph) const

  Compares `*this` *with the certificate for polyhedron* `ph`*.*

- int compare (const Polyhedron &ph) const

  Compares `*this` *with the certificate for polyhedron* `ph`*.*

Private Attributes

- dimension_type affine_dim

  Affine dimension of the polyhedron.

- dimension_type num_constraints

  Cardinality of a non-redundant constraint system for the polyhedron.

11.75.1 Detailed Description

A convergence certificate for the H79 widening operator. Convergence certificates are used to instantiate the BHZ03 framework so as to define widening operators for the finite powerset domain.

Note

The convergence of the H79 widening can also be certified by BHRZ03_Certificate.

Definition at line 41 of file H79_Certificate.defs.hh.

11.75.2 Constructor & Destructor Documentation

11.75.2.1 Parma_Polyhedra_Library::H79_Certificate::H79_Certificate () [inline]

Default constructor.

Definition at line 32 of file H79_Certificate.inlines.hh.

11.75.2.2 template<typename PH > Parma_Polyhedra_Library::H79_Certificate::H79_Certificate (const PH & *ph*) [inline]

Constructor: computes the certificate for `ph`.

Definition at line 56 of file H79_Certificate.inlines.hh.

References affine_dim, Parma_Polyhedra_Library::NECESSARILY_CLOSED, and num_constraints.

11.75.2.3 Parma_Polyhedra_Library::H79_Certificate::H79_Certificate (const Polyhedron & *ph*)

Constructor: computes the certificate for `ph`.

Definition at line 35 of file H79_Certificate.cc.

References affine_dim, Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::minimized_constraints(), num_constraints, and Parma_Polyhedra_Library::Polyhedron::space_dimension().

11.75.2.4 Parma_Polyhedra_Library::H79_Certificate::H79_Certificate (const H79_Certificate & *y*) [inline]

Copy constructor.

Definition at line 38 of file H79_Certificate.inlines.hh.

11.75.2.5 Parma_Polyhedra_Library::H79_Certificate::~H79_Certificate () [inline]

Destructor.

Definition at line 43 of file H79_Certificate.inlines.hh.

11.75.3 Member Function Documentation

11.75.3.1 int Parma_Polyhedra_Library::H79_Certificate::compare (const Polyhedron & *ph*) const

Compares `*this` with the certificate for polyhedron `ph`.

Definition at line 76 of file H79_Certificate.cc.

References affine_dim, Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::minimized_constraints(), num_constraints, and Parma_Polyhedra_Library::Polyhedron::space_dimension().

11.75.3.2 template<typename PH > int Parma_Polyhedra_Library::H79_Certificate::compare (const PH & *ph*) const [inline]

Compares `*this` with the certificate for polyhedron `ph`.

Definition at line 65 of file H79_Certificate.inlines.hh.

References compare(), and Parma_Polyhedra_Library::NECESSARILY_CLOSED.

11.75.3.3 int Parma_Polyhedra_Library::H79_Certificate::compare (const H79_Certificate & *y*) const

The comparison function for certificates.

Returns

-1, 0 or 1 depending on whether `*this` is smaller than, equal to, or greater than `y`, respectively.

Compares `*this` with `y`, using a total ordering which is a refinement of the limited growth ordering relation for the H79 widening.

Definition at line 66 of file H79_Certificate.cc.

References affine_dim, and num_constraints.

Referenced by compare(), and Parma_Polyhedra_Library::H79_Certificate::Compare::operator()().

11.75.4 Member Data Documentation

11.75.4.1 dimension_type Parma_Polyhedra_Library::H79_Certificate::affine_dim [private]

Affine dimension of the polyhedron.

Definition at line 91 of file H79_Certificate.defs.hh.

Referenced by compare(), and H79_Certificate().

11.75.4.2 dimension_type Parma_Polyhedra_Library::H79_Certificate::num_constraints [private]

Cardinality of a non-redundant constraint system for the polyhedron.

Definition at line 93 of file H79_Certificate.defs.hh.

Referenced by compare(), and H79_Certificate().

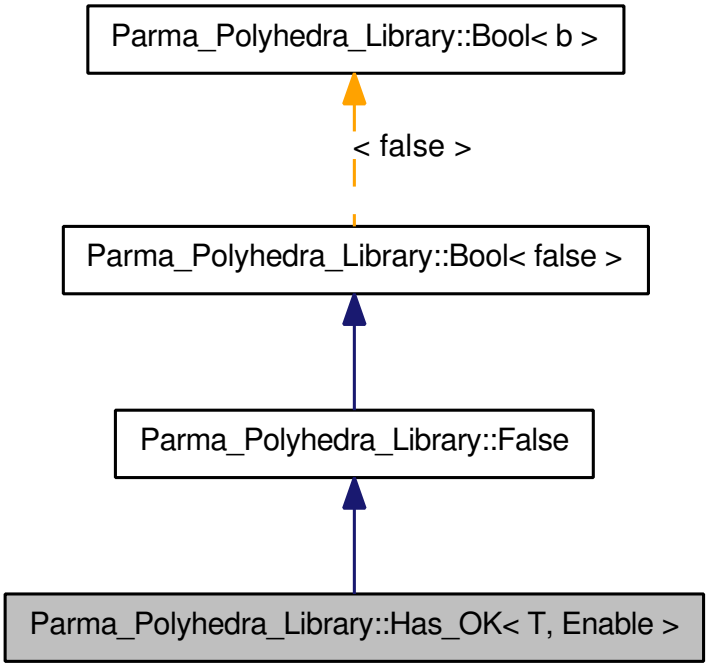
The documentation for this class was generated from the following files:

- H79_Certificate.defs.hh
- H79_Certificate.inlines.hh
- H79_Certificate.cc

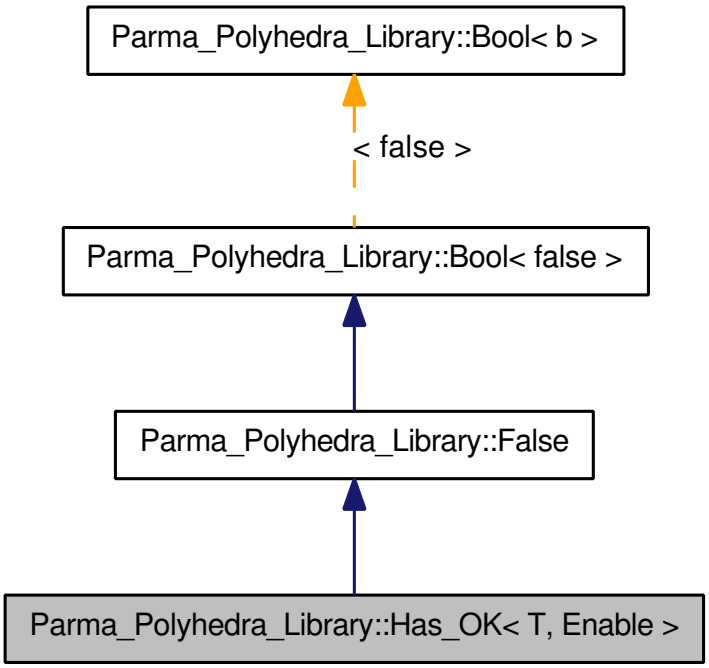
11.76 Parma_Polyhedra_Library::Has_OK< T, Enable > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Has_OK< T, Enable >:



Collaboration diagram for Parma_Polyhedra_Library::Has_OK< T, Enable >:



11.76.1 Detailed Description

template<typename T, typename Enable = void> struct Parma_Polyhedra_Library::Has_OK< T, Enable >

Definition at line 435 of file globals.defs.hh.

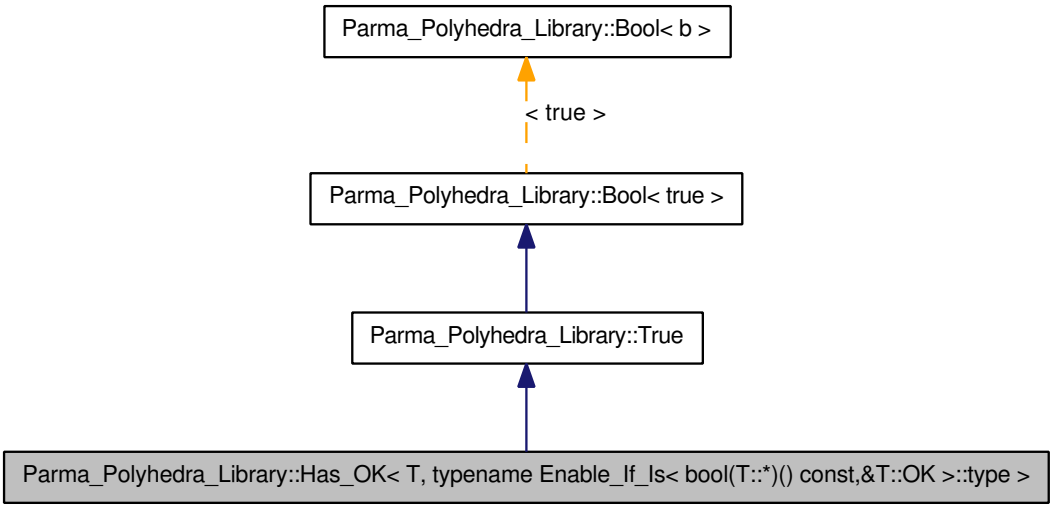
The documentation for this struct was generated from the following file:

- globals.defs.hh

11.77 Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::∗)() const,&T::OK >::type > Struct Template Reference

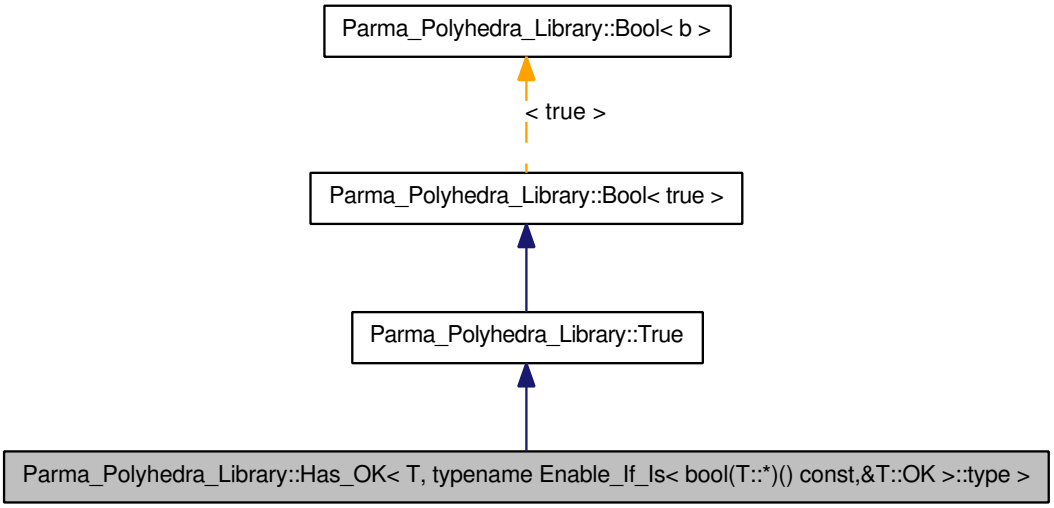
`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::∗)() const,&T::OK >::type >:



Collaboration diagram for Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::∗)()

const,&T::OK >::type >:



11.77.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::∗)() const,&T::OK >::type >

Definition at line 438 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

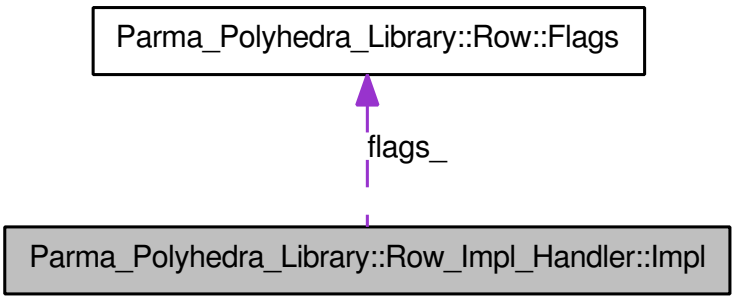
- globals.defs.hh

11.78 Parma_Polyhedra_Library::Row_Impl_Handler::Impl Class Reference

The actual implementation of a Row object.

`#include <Row.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Row_Impl_Handler::Impl:



Public Member Functions

- Impl (Row::Flags f)

Constructor.

- ~Impl ()
  Destructor.
- void expand_within_capacity (dimension_type new_size)
  Expands the row to size `new_size`.
- void shrink (dimension_type new_size)
  Shrinks the row by erasing elements at the end.
- void copy_construct_coefficients (const Impl &y)
  Exception-safe copy construction mechanism for coefficients.
- memory_size_type total_memory_in_bytes () const
  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.
- memory_size_type total_memory_in_bytes (dimension_type capacity) const
  Returns the total size in bytes of the memory occupied by `*this`.
- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.

Flags accessors

- const Row::Flags & flags () const
  Returns a const reference to the flags of `*this`.
- Row::Flags & flags ()
  Returns a non-const reference to the flags of `*this`.

Size accessors

- dimension_type size () const
  Returns the actual size of `this`.
- void set_size (dimension_type new_size)
  Sets to `new_size` the actual size of `*this`.
- void bump_size ()
  Increment the size of `*this` by 1.

Subscript operators

- Coefficient & operator[ ] (dimension_type k)
  Returns a reference to the element of `*this` indexed by `k`.
- Coefficient_traits::const_reference operator[ ] (dimension_type k) const
  Returns a constant reference to the element of `*this` indexed by `k`.

Static Public Member Functions

- static dimension_type max_size ()

  Returns the size() of the largest possible Impl.

Custom allocator and deallocator

- static void * operator new (size_t fixed_size, dimension_type capacity)

  Placement allocation function.

- static void operator delete (void *p)

  Usual (non-placement) deallocation function.

- static void operator delete (void *p, dimension_type capacity)

  Placement deallocation function.

Private Member Functions

- Impl ()

  Private and unimplemented: default construction is not allowed.

- Impl (const Impl &y)

  Private and unimplemented: copy construction is not allowed.

- Impl & operator= (const Impl &)

  Private and unimplemented: assignment is not allowed.

Private Attributes

- dimension_type size_

  The number of coefficients in the row.

- Row::Flags flags_

  The flags of this row.

- Coefficient vec_ [1]

  The vector of coefficients.

11.78.1 Detailed Description

The actual implementation of a Row object. The class Row_Impl_Handler::Impl provides the implementation of Row objects and, in particular, of the corresponding memory allocation functions.

Definition at line 389 of file Row.defs.hh.

11.78.2 Constructor & Destructor Documentation

11.78.2.1 Parma_Polyhedra_Library::Row_Impl_Handler::Impl::Impl (Row::Flags *f*) [inline]

Constructor.

Definition at line 117 of file Row.inlines.hh.

11.78.2.2 Parma_Polyhedra_Library::Row_Impl_Handler::Impl::~Impl () [inline]

Destructor.

Uses `shrink()` method with argument 0 to delete all the row elements.

Definition at line 122 of file Row.inlines.hh.

References shrink().

11.78.2.3 Parma_Polyhedra_Library::Row_Impl_Handler::Impl::Impl () [private]

Private and unimplemented: default construction is not allowed.

11.78.2.4 Parma_Polyhedra_Library::Row_Impl_Handler::Impl::Impl (const Impl & *y*) [private]

Private and unimplemented: copy construction is not allowed.

11.78.3 Member Function Documentation

11.78.3.1 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::bump_size () [inline]

Increment the size of `*this` by 1.

Definition at line 112 of file Row.inlines.hh.

References size_.

Referenced by copy_construct_coefficients().

11.78.3.2 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::copy_construct_coefficients (const Impl & *y*)

Exception-safe copy construction mechanism for coefficients.

Definition at line 66 of file Row.cc.

References bump_size(), size(), and vec_.

Referenced by Parma_Polyhedra_Library::Row::copy_construct_coefficients().

11.78.3.3 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::expand_within_capacity (dimension_type *new_size*)

Expands the row to size `new_size`.

It is assumed that `new_size` is between the current size and capacity.

Definition at line 35 of file Row.cc.

Referenced by Parma_Polyhedra_Library::Row::expand_within_capacity().

11.78.3.4 PPL::memory_size_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Definition at line 202 of file Row.cc.

References Parma_Polyhedra_Library::external_memory_in_bytes(), size(), and vec_.

Referenced by total_memory_in_bytes().

11.78.3.5 Row::Flags & Parma_Polyhedra_Library::Row_Impl_Handler::Impl::flags () [inline]

Returns a non-const reference to the flags of `*this`.

Definition at line 132 of file Row.inlines.hh.

References flags_.

11.78.3.6 const Row::Flags & Parma_Polyhedra_Library::Row_Impl_Handler::Impl::flags () const [inline]

Returns a const reference to the flags of `*this`.

Definition at line 127 of file Row.inlines.hh.

References flags_.

Referenced by Parma_Polyhedra_Library::Row::flags().

11.78.3.7 dimension_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::max_size () [inline, static]

Returns the size() of the largest possible Impl.

Definition at line 97 of file Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Row::max_size().

11.78.3.8 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator delete (void * *p*, dimension_type *capacity*) [inline, static]

Placement deallocation function.

Uses the standard operator delete to free the memory `p` points to.

Definition at line 92 of file Row.inlines.hh.

11.78.3.9 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator delete (void * *p*) [inline, static]

Usual (non-placement) deallocation function.

Uses the standard delete operator to free the memory `p` points to.

Note

The definition of this custom deallocation function is required since otherwise the placement deallocation function `static void operator delete(void* p, dimension_type capacity);` would be wrongly interpreted as a usual (non-placement) deallocation function (see C++98 3.7.3.2p2). This happens because `dimension_type` is just an alias for `std::size_t`. See also http://gcc.gnu.org/bugzilla/show_bug.cgi?id=42115

Definition at line 87 of file Row.inlines.hh.

11.78.3.10 void * Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator new (size_t *fixed_size*, dimension_type *capacity*) [inline, static]

Placement allocation function.

Allocates a chunk of memory able to contain `capacity` Coefficient objects beyond the specified `fixed_size` and returns a pointer to the newly allocated memory.

Definition at line 76 of file Row.inlines.hh.

11.78.3.11 Impl& Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator= (const Impl &) [private]

Private and unimplemented: assignment is not allowed.

11.78.3.12 Coefficient_traits::const_reference Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the element of `*this` indexed by `k`.

Definition at line 143 of file Row.inlines.hh.

References size(), and vec_.

11.78.3.13 Coefficient & Parma_Polyhedra_Library::Row_Impl_Handler::Impl::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the element of `*this` indexed by `k`.

Definition at line 137 of file Row.inlines.hh.

References size(), and vec_.

11.78.3.14 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::set_size (dimension_type *new_size*) [inline]

Sets to `new_size` the actual size of `*this`.

Definition at line 107 of file Row.inlines.hh.

References size_.

Referenced by shrink().

11.78.3.15 void Parma_Polyhedra_Library::Row_Impl_Handler::Impl::shrink (dimension_type *new_size*)

Shrinks the row by erasing elements at the end.

It is assumed that `new_size` is not greater than the current size.

Definition at line 49 of file Row.cc.

References set_size(), size(), and vec_.

Referenced by Parma_Polyhedra_Library::Row::shrink(), and ~Impl().

11.78.3.16 dimension_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::size () const [inline]

Returns the actual size of `this`.

Definition at line 102 of file Row.inlines.hh.

References size_.

Referenced by copy_construct_coefficients(), external_memory_in_bytes(), operator[ ](), shrink(), and Parma_Polyhedra_Library::Row::size().

11.78.3.17 memory_size_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::total_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 149 of file Row.inlines.hh.

References external_memory_in_bytes().

11.78.3.18 memory_size_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Definition at line 160 of file Row.inlines.hh.

References size_.

Referenced by Parma_Polyhedra_Library::Row::external_memory_in_bytes().

11.78.4 Member Data Documentation

11.78.4.1 Row::Flags Parma_Polyhedra_Library::Row_Impl_Handler::Impl::flags_ [private]

The flags of this row.

Definition at line 501 of file Row.defs.hh.

Referenced by flags().

11.78.4.2 dimension_type Parma_Polyhedra_Library::Row_Impl_Handler::Impl::size_ [private]

The number of coefficients in the row.

Definition at line 498 of file Row.defs.hh.

Referenced by bump_size(), set_size(), size(), and total_memory_in_bytes().

11.78.4.3 Coefficient Parma_Polyhedra_Library::Row_Impl_Handler::Impl::vec_[1] [private]

The vector of coefficients.

Definition at line 508 of file Row.defs.hh.

Referenced by copy_construct_coefficients(), external_memory_in_bytes(), operator[ ](), and shrink().

The documentation for this class was generated from the following files:

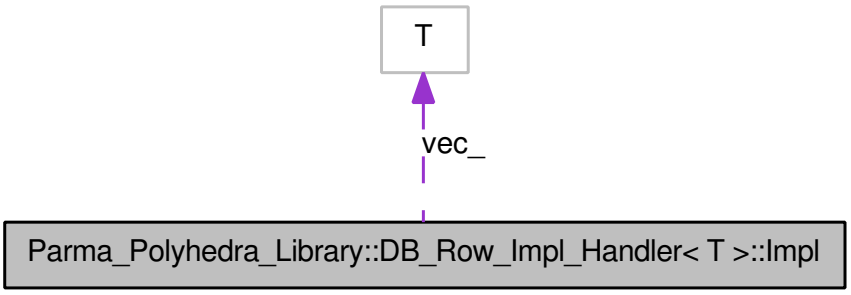
- Row.defs.hh
- Row.inlines.hh
- Row.cc

11.79 Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl Class Reference

The real implementation of a DB_Row object.

`#include <DB_Row.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl:



Public Member Functions

- Impl ()

  Default constructor.

- ~Impl ()

  Destructor.

- void expand_within_capacity (dimension_type new_size)

  Expands the row to size `new_size`.

- void shrink (dimension_type new_size)

  Shrinks the row by erasing elements at the end.

- void copy_construct_coefficients (const Impl &y)

  Exception-safe copy construction mechanism for coefficients.

- template<typename U >
  void construct_upward_approximation (const U &y)
  Exception-safe upward approximation construction mechanism for coefficients.

- memory_size_type total_memory_in_bytes () const
  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type total_memory_in_bytes (dimension_type capacity) const
  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.

Size accessors.

- dimension_type size () const
  Returns the actual size of `this`.

- void set_size (dimension_type new_sz)
  Sets to `new_sz` *the actual size of* `*this`.

- void bump_size ()
  Increments the size of `*this` *by 1.*

Subscript operators.

- T & operator[ ] (dimension_type k)
  Returns a reference to the element of `*this` *indexed by* `k`.

- const T & operator[ ] (dimension_type k) const
  Returns a constant reference to the element of `*this` *indexed by* `k`.

Static Public Member Functions

- static dimension_type max_size ()
  Returns the size() of the largest possible Impl.

Custom allocator and deallocator.

- static void * operator new (size_t fixed_size, dimension_type capacity)
  Allocates a chunk of memory able to contain `capacity` *T objects beyond the specified* `fixed_size` *and returns a pointer to the new allocated memory.*

- static void operator delete (void *p)
  Uses the standard delete operator to free the memory `p` *points to.*

- static void operator delete (void *p, dimension_type capacity)
  Placement version: uses the standard operator delete to free the memory `p` *points to.*

Private Member Functions

- Impl (const Impl &y)

  Private and unimplemented: copy construction is not allowed.

- Impl & operator= (const Impl &)

  Private and unimplemented: assignment is not allowed.

- void copy_construct (const Impl &y)

  Exception-safe copy construction mechanism.

Private Attributes

- dimension_type size_

  The number of coefficients in the row.

- T vec_ [1]

  The vector of coefficients.

Friends

- class DB_Row< T >

11.79.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl

The real implementation of a DB_Row object. The class DB_Row_Impl_Handler::Impl provides the implementation of DB_Row objects and, in particular, of the corresponding memory allocation functions.

Definition at line 337 of file DB_Row.defs.hh.

11.79.2 Constructor & Destructor Documentation

11.79.2.1 template<typename T > Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::Impl () [inline]

Default constructor.

Definition at line 107 of file DB_Row.inlines.hh.

11.79.2.2 template<typename T > Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::~Impl () [inline]

Destructor.

Uses shrink() method with argument 0 to delete all the row elements.

Definition at line 113 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::shrink().

11.79.2.3 template<typename T> Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::Impl (const Impl & *y*) [private]

Private and unimplemented: copy construction is not allowed.

11.79.3 Member Function Documentation

11.79.3.1 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::bump_size () [inline]

Increments the size of `*this` by 1.

Definition at line 101 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::construct_upward_approximation(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::copy_construct_coefficients(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity().

11.79.3.2 template<typename T > template<typename U > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::construct_upward_approximation (const U & *y*) [inline]

Exception-safe upward approximation construction mechanism for coefficients.

Definition at line 34 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::bump_size(), Parma_Polyhedra_Library::construct(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::construct_upward_approximation().

11.79.3.3 template<typename T> void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::copy_construct (const Impl & *y*) [private]

Exception-safe copy construction mechanism.

11.79.3.4 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::copy_construct_coefficients (const Impl & *y*) [inline]

Exception-safe copy construction mechanism for coefficients.

Definition at line 94 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::bump_size(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::copy_construct_coefficients().

11.79.3.5 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity (dimension_type *new_size*) [inline]

Expands the row to size `new_size`.

It is assumed that `new_size` is between the current size and capacity.

Definition at line 58 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::bump_size(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::max_size(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::expand_within_capacity().

11.79.3.6 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 117 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes().

11.79.3.7 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::max_size () [inline, static]

Returns the size() of the largest possible Impl.

Definition at line 83 of file DB_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity().

11.79.3.8 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator delete (void * *p*, dimension_type *capacity*) [inline, static]

Placement version: uses the standard operator delete to free the memory `p` points to.

Definition at line 56 of file DB_Row.inlines.hh.

11.79.3.9 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator delete (void * *p*) [inline, static]

Uses the standard delete operator to free the memory `p` points to.

Definition at line 50 of file DB_Row.inlines.hh.

11.79.3.10 template<typename T > void * Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator new (size_t *fixed_size*, dimension_type *capacity*) [inline, static]

Allocates a chunk of memory able to contain `capacity` T objects beyond the specified `fixed_size` and returns a pointer to the new allocated memory.

Definition at line 38 of file DB_Row.inlines.hh.

11.79.3.11 template<typename T> Impl& Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator= (const Impl &) [private]

Private and unimplemented: assignment is not allowed.

11.79.3.12 template<typename T > const T & Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the element of `*this` indexed by `k`.

Definition at line 141 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.79.3.13 template<typename T > T & Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the element of `*this` indexed by `k`.

Definition at line 134 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

11.79.3.14 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::set_size (dimension_type *new_sz*) [inline]

Sets to `new_sz` the actual size of `*this`.

Definition at line 95 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::shrink().

11.79.3.15 template<typename T > void Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::shrink (dimension_type *new_size*) [inline]

Shrinks the row by erasing elements at the end.

It is assumed that `new_size` is not greater than the current size.

Definition at line 76 of file DB_Row.templates.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::set_size(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::shrink(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::~Impl().

11.79.3.16 template<typename T > dimension_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size () const [inline]

Returns the actual size of `this`.

Definition at line 89 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::external_memory_in_bytes(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator[ ](), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::shrink(), and Parma_Polyhedra_Library::DB_Row< T >::size().

11.79.3.17 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 63 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::external_memory_in_bytes().

11.79.3.18 template<typename T > memory_size_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Definition at line 75 of file DB_Row.inlines.hh.

References Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::external_memory_in_bytes().

11.79.4 Friends And Related Function Documentation

11.79.4.1 template<typename T> friend class DB_Row< T > [friend]

Definition at line 428 of file DB_Row.defs.hh.

11.79.5 Member Data Documentation

11.79.5.1 template<typename T> dimension_type Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size_ [private]

The number of coefficients in the row.

Definition at line 431 of file DB_Row.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::bump_size(), Parma_Polyhedra_Library::DB_Row< T >::end(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::set_size(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::size(), and Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::total_memory_in_bytes().

11.79.5.2 template<typename T> T Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::vec_[1] [private]

The vector of coefficients.

Definition at line 438 of file DB_Row.defs.hh.

Referenced by Parma_Polyhedra_Library::DB_Row< T >::begin(), Parma_Polyhedra_Library::DB_-Row_Impl_Handler< T >::Impl::construct_upward_approximation(), Parma_Polyhedra_Library::DB_-Row_Impl_Handler< T >::Impl::copy_construct_coefficients(), Parma_Polyhedra_Library::DB_Row< T >::end(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::expand_within_capacity(), Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::external_memory_in_bytes(), Parma_-Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl::operator[ ](), and Parma_Polyhedra_-Library::DB_Row_Impl_Handler< T >::Impl::shrink().

The documentation for this class was generated from the following files:

- DB_Row.defs.hh
- DB_Row.inlines.hh
- DB_Row.templates.hh

11.80 Parma_Polyhedra_Library::Init Class Reference

Class for initialization and finalization.

`#include <Init.defs.hh>`

Public Member Functions

- Init ()

  Initializes the PPL.

- ~Init ()

  Finalizes the PPL.

Static Private Attributes

- static unsigned int count = 0

  Count the number of objects created.

- static fpu_rounding_direction_type old_rounding_direction

Friends

- void set_rounding_for_PPL ()

  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

- void restore_pre_PPL_rounding ()

  Sets the FPU rounding mode as it was before initialization of the PPL.

11.80.1 Detailed Description

Class for initialization and finalization. *Nifty Counter* initialization class, ensuring that the library is initialized only once and before its first use. A count of the number of translation units using the library is maintained. A static object of Init type will be declared by each translation unit using the library. As a result, only one of them will initialize and properly finalize the library.

Definition at line 67 of file Init.defs.hh.

11.80.2 Constructor & Destructor Documentation

11.80.2.1 Parma_Polyhedra_Library::Init::Init ()

Initializes the PPL.

Definition at line 137 of file Init.cc.

References Parma_Polyhedra_Library::Coefficient_constants_initialize(), count, Parma_Polyhedra_Library::Variable::default_output_function(), Parma_Polyhedra_Library::initialize(), old_rounding_direction, ppl_set_GMP_memory_allocation_functions(), Parma_Polyhedra_Library::set_irrational_precision(), and Parma_Polyhedra_Library::Variable::set_output_function().

11.80.2.2 Parma_Polyhedra_Library::Init::~Init ()

Finalizes the PPL.

Definition at line 183 of file Init.cc.

References Parma_Polyhedra_Library::Coefficient_constants_finalize(), count, Parma_Polyhedra_Library::finalize(), and old_rounding_direction.

11.80.3 Friends And Related Function Documentation

11.80.3.1 void restore_pre_PPL_rounding () [friend]

Sets the FPU rounding mode as it was before initialization of the PPL.

After calling this function it is absolutely necessary to call set_rounding_for_PPL() before using any PPL abstractions based on floating point numbers. This is performed automatically at finalization-time.

Definition at line 40 of file Init.inlines.hh.

11.80.3.2 void set_rounding_for_PPL () [friend]

Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

This is performed automatically at initialization-time. Calling this function is needed only if restore_pre_PPL_rounding() has been previously called.

Definition at line 33 of file Init.inlines.hh.

11.80.4 Member Data Documentation

11.80.4.1 unsigned int Parma_Polyhedra_Library::Init::count = 0 [static, private]

Count the number of objects created.

Definition at line 77 of file Init.defs.hh.

Referenced by Init(), and ∼Init().

11.80.4.2 PPL::fpu_rounding_direction_type Parma_Polyhedra_Library::Init::old_rounding_direction [static, private]

Definition at line 78 of file Init.defs.hh.

Referenced by Init(), Parma_Polyhedra_Library::restore_pre_PPL_rounding(), and ∼Init().

The documentation for this class was generated from the following files:

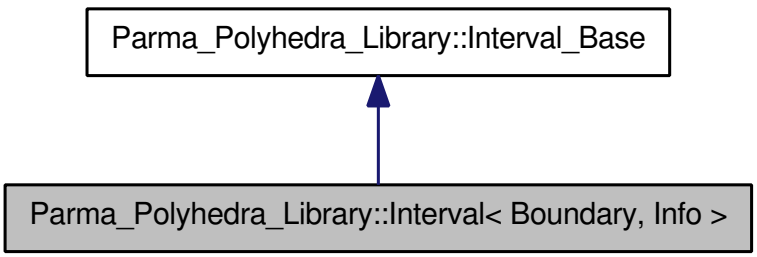
- Init.defs.hh
- Init.cc

11.81 Parma_Polyhedra_Library::Interval< Boundary, Info > Class Template Reference

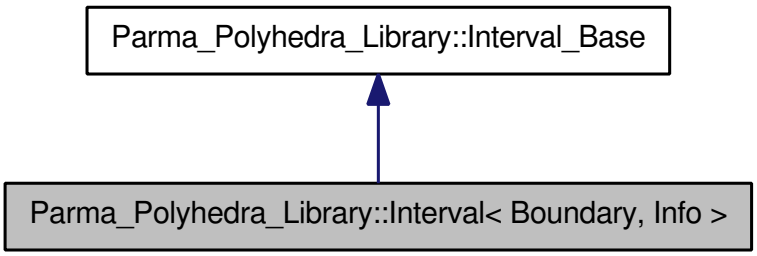
A generic, not necessarily closed, possibly restricted interval.

`#include <Interval.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval< Boundary, Info >:



Collaboration diagram for Parma_Polyhedra_Library::Interval< Boundary, Info >:



Public Types

- typedef Boundary boundary_type
- typedef Info info_type
- typedef Interval_NS::Property Property

Public Member Functions

- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, Interval & >::type operator= (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, Interval & >::type operator+= (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, Interval & >::type operator-= (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, Interval & >::type operator*= (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, Interval & >::type operator/= (const T &x)
- void swap (Interval &y)

  Swaps `*this` *with* `y`.

- Info & info ()
- const Info & info () const
- Boundary & lower ()
- const Boundary & lower () const
- Boundary & upper ()
- const Boundary & upper () const
- I_Constraint< boundary_type > lower_constraint () const
- I_Constraint< boundary_type > upper_constraint () const
- bool has_restriction () const
- I_Result normalize () const
- bool is_empty () const
- bool check_empty (I_Result r) const

- bool is_singleton () const
- bool lower_is_open () const
- bool upper_is_open () const
- bool lower_is_boundary_infinity () const
- bool upper_is_boundary_infinity () const
- bool lower_is_domain_inf () const
- bool upper_is_domain_sup () const
- bool is_bounded () const
- bool is_universe () const
- I_Result lower_extend ()
- template<typename C >
  Enable_If< Is_Same_Or_Derived< I_Constraint_Base, C >::value, I_Result >::type lower_extend (const C &c)
- I_Result upper_extend ()
- template<typename C >
  Enable_If< Is_Same_Or_Derived< I_Constraint_Base, C >::value, I_Result >::type upper_extend (const C &c)
- I_Result build ()
- template<typename C >
  Enable_If< Is_Same_Or_Derived< I_Constraint_Base, C >::value, I_Result >::type build (const C &c)
- template<typename C1 , typename C2 >
  Enable_If< Is_Same_Or_Derived< I_Constraint_Base, C1 >::value &&Is_Same_Or_Derived< I_Constraint_Base, C2 >::value, I_Result >::type build (const C1 &c1, const C2 &c2)
- template<typename C >
  Enable_If< Is_Same_Or_Derived< I_Constraint_Base, C >::value, I_Result >::type add_constraint (const C &c)
- I_Result assign (Degenerate_Element e)
- template<typename From >
  Enable_If< Is_Special< From >::value, I_Result >::type assign (const From &)
- I_Result set_infinities ()
- bool is_topologically_closed () const
- void topological_closure_assign ()

  Assigns to `*this` *its topological closure.*

- void remove_inf ()
- void remove_sup ()
- bool is_infinity () const
- bool contains_integer_point () const
- void drop_some_non_integer_points ()
- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type wrap_assign (Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, const From &refinement)
- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void ascii_dump (std::ostream &s) const

- bool ascii_load (std::istream &s)
- bool OK () const
- Interval ()
- template<typename T >
  Interval (const T &x)
- Interval (const char *s)

  Builds the smallest interval containing the number whose textual representation is contained in `s`.

- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type contains (const T &y) const
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type strictly_contains (const T &y) const
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type is_disjoint_from (const T &y) const
- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type assign (const From &x)
- template<typename Type >
  Enable_If< Is_Singleton< Type >::value||Is_Interval< Type >::value, bool >::type can_be_exactly_joined_to (const Type &x) const
- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type join_assign (const From &x)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type join_assign (const From1 &x, const From2 &y)
- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type intersect_assign (const From &x)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type intersect_assign (const From1 &x, const From2 &y)
- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type difference_assign (const From &x)

  Assigns to `*this` *the smallest interval containing the set-theoretic difference of* `*this` *and* `x`.

- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type difference_assign (const From1 &x, const From2 &y)

  Assigns to `*this` *the smallest interval containing the set-theoretic difference of* `x` *and* `y`.

- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type lower_approximation_difference_assign (const From &x)

  Assigns to `*this` *the largest interval contained in the set-theoretic difference of* `*this` *and* `x`.

- template<typename From >
  Enable_If< Is_Interval< From >::value, bool >::type simplify_using_context_assign (const From &y)

  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`.

- template<typename From >
  Enable_If< Is_Interval< From >::value, void >::type empty_intersection_assign (const From &y)

  Assigns to `*this` *an interval having empty intersection with* `y`*. The assigned interval should be as large as possible.*

- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type refine_existential (Relation_Symbol rel, const From &x)

  Refines `to` *according to the existential relation* `rel` *with* `x`.

- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type refine_universal (Relation_Symbol rel, const From &x)

  Refines `to` *so that it satisfies the universal relation* `rel` *with* `x`.

- template<typename From >
  Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type neg_assign (const From &x)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type add_assign (const From1 &x, const From2 &y)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type sub_assign (const From1 &x, const From2 &y)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type mul_assign (const From1 &x, const From2 &y)
- template<typename From1 , typename From2 >
  Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type div_assign (const From1 &x, const From2 &y)
- template<typename From , typename Iterator >
  Enable_If< Is_Interval< From >::value, void >::type CC76_widening_assign (const From &y, Iterator first, Iterator last)

Static Public Member Functions

- static bool is_always_topologically_closed ()

Private Member Functions

- PPL_COMPILE_TIME_CHECK (!Info::store_special||!std::numeric_limits< Boundary >::has_infinity,"store_special is meaningless"" when boundary type may contains infinity")

- Info & w_info () const
- Result lower_normalize () const
- Result upper_normalize () const

Private Attributes

- Boundary lower_
- Boundary upper_

Related Functions

(Note that these are not member functions.)

- template<typename Boundary , typename Info >
  void swap (Parma_Polyhedra_Library::Interval< Boundary, Info > &x, Parma_Polyhedra_Library::Interval< Boundary, Info > &y)

11.81.1 Detailed Description

template<typename Boundary, typename Info> class Parma_Polyhedra_Library::Interval< Boundary, Info >

A generic, not necessarily closed, possibly restricted interval. The class template type parameter `Boundary` represents the type of the interval boundaries, and can be chosen, among other possibilities, within one of the following number families:

- a bounded precision native integer type (that is, from `signed char` to `long long` and from `int8_t` to `int64_t`);
- a bounded precision floating point type (`float`, `double` or `long double`);
- an unbounded integer or rational type, as provided by the C++ interface of GMP (`mpz_class` or `mpq_class`).

The class template type parameter `Info` allows to control a number of features of the class, among which:

- the ability to support open as well as closed boundaries;
- the ability to represent empty intervals in addition to nonempty ones;
- the ability to represent intervals of extended number families that contain positive and negative infinities;
- the ability to support (independently from the type of the boundaries) plain intervals of real numbers and intervals subject to generic *restrictions* (e.g., intervals of integer numbers).

Definition at line 90 of file Interval.defs.hh.

11.81.2 Member Typedef Documentation

11.81.2.1 template<typename Boundary, typename Info> typedef Boundary Parma_Polyhedra_Library::Interval< Boundary, Info >::boundary_type

Definition at line 144 of file Interval.defs.hh.

11.81.2.2 template<typename Boundary, typename Info> typedef Info Parma_Polyhedra_Library::Interval< Boundary, Info >::info_type

Definition at line 145 of file Interval.defs.hh.

11.81.2.3 template<typename Boundary, typename Info> typedef Interval_NS::Property Parma_Polyhedra_Library::Interval< Boundary, Info >::Property

Definition at line 147 of file Interval.defs.hh.

11.81.3 Constructor & Destructor Documentation

11.81.3.1 template<typename Boundary, typename Info> Parma_Polyhedra_Library::Interval< Boundary, Info >::Interval () [inline]

Definition at line 652 of file Interval.defs.hh.

11.81.3.2 template<typename Boundary, typename Info> template<typename T > Parma_Polyhedra_Library::Interval< Boundary, Info >::Interval (const T & *x*) [inline, explicit]

Definition at line 656 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign().

11.81.3.3 template<typename Boundary, typename Info> Parma_Polyhedra_Library::Interval< Boundary, Info >::Interval (const char * *s*) [explicit]

Builds the smallest interval containing the number whose textual representation is contained in `s`.

11.81.4 Member Function Documentation

11.81.4.1 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::add_assign (const From1 & *x*, const From2 & *y*) [inline]

Definition at line 617 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::is_infinity(), Parma_Polyhedra_Library::MINUS_INFINITY, Parma_Polyhedra_Library::PLUS_INFINITY, and Parma_Polyhedra_Library::Checked::Result.

Referenced by Parma_Polyhedra_Library::operator+(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::operator+=().

11.81.4.2 template<typename Boundary, typename Info> template<typename C > Enable_If<Is_Same_Or_Derived<I_Constraint_Base, C>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::add_constraint (const C & *c*) [inline]

Definition at line 368 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::build(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::build().

11.81.4.3 template<typename Boundary , typename Info > void Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_dump (std::ostream & *s*) const [inline]

Definition at line 1092 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::ascii_dump(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.4 template<typename Boundary , typename Info > bool Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load (std::istream & *s*) [inline]

Definition at line 1105 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.5 template<typename Boundary, typename Info> template<typename From > Enable_If<Is_Singleton<From>::value || Is_Interval<From>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::assign (const From & *x*) [inline]

11.81.4.6 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::assign (const From & *x*) [inline]

Definition at line 398 of file Interval.defs.hh.

References Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.7 template<typename Boundary, typename Info> I_Result Parma_Polyhedra_Library::Interval< Boundary, Info >::assign (Degenerate_Element *e*) [inline]

Definition at line 374 of file Interval.defs.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_, Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::UNIVERSE, Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::build(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::Interval(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::operator=(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.8 template<typename Boundary, typename Info> template<typename C1 , typename C2 > Enable_If<Is_Same_Or_Derived<I_Constraint_Base, C1>::value && Is_Same_Or_Derived<I_Constraint_Base, C2>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::build (const C1 & *c1*, const C2 & *c2*) [inline]

Definition at line 344 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::add_constraint(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::build(), and Parma_Polyhedra_Library::EMPTY.

11.81.4.9 template<typename Boundary, typename Info> template<typename C > Enable_If<Is_Same_Or_Derived<I_Constraint_Base, C>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::build (const C & *c*) [inline]

Definition at line 318 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_existential(), and Parma_Polyhedra_Library::UNIVERSE.

11.81.4.10 template<typename Boundary, typename Info> I_Result Parma_Polyhedra_Library::Interval< Boundary, Info >::build () [inline]

Definition at line 312 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), and Parma_Polyhedra_Library::UNIVERSE.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_constraint(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::build().

11.81.4.11 template<typename Boundary , typename Info > template<typename Type > Enable_If< Is_Singleton< Type >::value||Is_Interval< Type >::value, bool >::type Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to (const Type & *x*) const [inline]

Definition at line 317 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Checked::eq, Parma_Polyhedra_Library::eq_restriction(), Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Checked::lt, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.12 template<typename Boundary, typename Info> template<typename From , typename Iterator > Enable_If<Is_Interval<From>::value, void>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::CC76_widening_assign (const From & *y*, Iterator *first*, Iterator *last*) [inline]

11.81.4.13 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::check_empty (I_Result *r*) const [inline]

Definition at line 242 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty().

11.81.4.14 template<typename Boundary , typename Info > template<typename T > Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type Parma_Polyhedra_Library::Interval< Boundary, Info >::contains (const T & *y*) const [inline]

Definition at line 190 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::contains_restriction(), Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.15 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point () const [inline]

Definition at line 474 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_bounded(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.16 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::difference_assign (const From1 & *x*, const From2 & *y*) [inline]

Assigns to `*this` the smallest interval containing the set-theoretic difference of `x` and `y`.

Definition at line 414 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Checked::lt, and Parma_Polyhedra_Library::Checked::Result.

11.81.4.17 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::difference_assign (const From & *x*) [inline]

Assigns to `*this` the smallest interval containing the set-theoretic difference of `*this` and `x`.

Definition at line 382 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Checked::lt, Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.18 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::div_assign (const From1 & *x*, const From2 & *y*) [inline]

+----------+----------+----------+ | / | yu < 0 | yl > 0 | +----------+----------+----------+ | xu<=0 |xu/yl,xl/yu|xl/yl,xu/yu| +----------+----------+----------+ |xl<=0 xu>=0|xu/yu,xl/yu|xl/yl,xu/yl| +----------+----------+----------+ | xl>=0 |xu/yu,xl/yl|xl/yu,xu/yl| +----------+----------+----------+

Definition at line 876 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::div_restriction(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::is_infinity(), Parma_Polyhedra_Library::MINUS_INFINITY, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::UNIVERSE.

Referenced by Parma_Polyhedra_Library::operator/(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::operator/=().

11.81.4.19 template<typename Boundary, typename Info> void Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points () [inline]

Definition at line 497 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.20 template<typename Boundary, typename Info> template<typename From > Enable_If<Is_Interval<From>::value, void>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::empty_intersection_assign (const From & *y*) [inline]

Assigns to `*this` an interval having empty intersection with `y`. The assigned interval should be as large as possible.

Note

Depending on interval restrictions, there could be many maximal intervals all inconsistent with respect to `y`.

11.81.4.21 template<typename Boundary , typename Info > memory_size_type Parma_Polyhedra_Library::Interval< Boundary, Info >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 42 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::total_memory_in_bytes().

11.81.4.22 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::has_restriction () const [inline]

Definition at line 221 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::is_universe(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize().

11.81.4.23 template<typename Boundary, typename Info> const Info& Parma_Polyhedra_Library::Interval< Boundary, Info >::info () const [inline]

Definition at line 188 of file Interval.defs.hh.

11.81.4.24 template<typename Boundary, typename Info> Info& Parma_Polyhedra_Library::Interval< Boundary, Info >::info () [inline]

Definition at line 184 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_dump(), Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::has_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_singleton(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_constraint(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_extend(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_domain_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::remove_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::remove_sup(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::swap(), Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_constraint(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_extend(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_domain_sup(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.25 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::intersect_assign (const From1 & *x*, const From2 & *y*) [inline]

Definition at line 358 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(),

Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_-Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::intersect_restriction(), Parma_-Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::min_assign(), and Parma_Polyhedra_-Library::Checked::Result.

11.81.4.26 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::intersect_assign (const From & *x*) [inline]

Definition at line 341 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_-Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_-NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::intersect_-restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_-Library::max_assign(), Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_-Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::add_constraint(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), and Parma_Polyhedra_-Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.27 template<typename Boundary, typename Info> static bool Parma_Polyhedra_-Library::Interval< Boundary, Info >::is_always_topologically_closed () [inline, static]

Definition at line 428 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed().

11.81.4.28 template<typename Boundary, typename Info> bool Parma_-Polyhedra_Library::Interval< Boundary, Info >::is_bounded () const [inline]

Definition at line 281 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point().

11.81.4.29 template<typename Boundary , typename Info > template<typename T > Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from (const T & *y*) const [inline]

Definition at line 232 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::Interval_-NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_-Library::Checked::lt, Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_-Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.30 template<typename Boundary, typename Info> bool Parma_-Polyhedra_Library::Interval< Boundary, Info >::is_empty () const [inline]

Definition at line 238 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Checked::lt, and Parma_-Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::check_empty(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::f_-is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_constraint(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::remove_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::remove_sup(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::upper_constraint(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.31 template<typename Boundary, typename Info> bool Parma_-Polyhedra_Library::Interval< Boundary, Info >::is_infinity () const [inline]

Definition at line 464 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::is_infinity().

11.81.4.32 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::is_singleton () const [inline]

Definition at line 247 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Checked::eq, Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::f_is_singleton().

11.81.4.33 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed () const [inline]

Definition at line 432 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::is_always_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.34 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::is_universe () const [inline]

Definition at line 286 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::has_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_domain_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_domain_sup().

11.81.4.35 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::join_assign (const From1 & *x*, const From2 & *y*) [inline]

Definition at line 290 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_

Library::max_assign(), Parma_Polyhedra_Library::min_assign(), and Parma_Polyhedra_Library::Checked::Result.

11.81.4.36 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::join_assign (const From & *x*) [inline]

Definition at line 269 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.37 template<typename Boundary, typename Info> const Boundary& Parma_Polyhedra_Library::Interval< Boundary, Info >::lower () const [inline]

Definition at line 196 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_.

11.81.4.38 template<typename Boundary, typename Info> Boundary& Parma_Polyhedra_Library::Interval< Boundary, Info >::lower () [inline]

Definition at line 192 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_dump(), Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::external_memory_in_bytes(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_singleton(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_constraint(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_extend(), Parma_Polyhedra_Library::Interval< Bound-

ary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_domain_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::swap(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.39 template<typename Boundary, typename Info> template<typename From > Enable_If<Is_Singleton<From>::value || Is_Interval<From>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_approximation_difference_assign (const From & *x*) [inline]

Assigns to `*this` the largest interval contained in the set-theoretic difference of `*this` and `x`.

11.81.4.40 template<typename Boundary, typename Info> I_Constraint<boundary_type> Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_constraint () const [inline]

Definition at line 208 of file Interval.defs.hh.

References Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open().

11.81.4.41 template<typename Boundary, typename Info> template<typename C > Enable_If<Is_Same_Or_Derived<I_Constraint_Base, C>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_extend (const C & *c*) [inline]

11.81.4.42 template<typename Boundary, typename Info> I_Result Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_extend () [inline]

Definition at line 292 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::lower().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.43 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity () const [inline]

Definition at line 261 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::OK().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_bounded(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.44 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_domain_inf () const [inline]

Definition at line 271 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::OK().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::is_universe().

11.81.4.45 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open () const [inline]

Definition at line 251 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::OK().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_constraint(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign().

11.81.4.46 template<typename Boundary, typename Info> Result Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize () const [inline, private]

Definition at line 99 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::w_info().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize().

11.81.4.47 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::mul_assign (const From1 & *x*, const From2 & *y*) [inline]

+---------+-----------+-----------+-----------------+ | ∗ | yl > 0 | yu < 0 | yl < 0, yu > 0 | +---------+-----------+-----------+-----------------+ | xl > 0 |xl∗yl,xu∗yu|xu∗yl,xl∗yu| xu∗yl,xu∗yu | +---------+-----------+-----------+-----------------+ | xu < 0 |xl∗yu,xu∗yl|xu∗yu,xl∗yl| xl∗yu,xl∗yl | +---------+-----------+-----------+-----------------+ |xl<0 xu>0|xl∗yu,xu∗yu|xu∗yl,xl∗yl|min(xl∗yu,xu∗yl),| | | |max(xl∗yl,xu∗yu) | +---------+-----------+-----------+-----------------+

Definition at line 707 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::is_infinity(), Parma_Polyhedra_Library::Checked::lt, Parma_Polyhedra_Library::MINUS_INFINITY, Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::PLUS_INFINITY, and Parma_Polyhedra_Library::Checked::Result.

Referenced by Parma_Polyhedra_Library::operator∗(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::operator∗=().

11.81.4.48 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::neg_assign (const From & *x*) [inline]

Definition at line 593 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::neg_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.49 template<typename Boundary, typename Info> I_Result Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize () const [inline]

Definition at line 225 of file Interval.defs.hh.

References Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::Interval< Boundary, Info >::has_restriction(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

11.81.4.50 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::OK () const [inline]

Definition at line 569 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::is_not_a_number(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_bounded(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_universe(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_domain_inf(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_domain_sup(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.51 template<typename Boundary, typename Info> template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, Interval&>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::operator*= (const T & *x*) [inline]

Definition at line 170 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::mul_assign().

11.81.4.52 template<typename Boundary, typename Info> template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, Interval&>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::operator+= (const T & *x*) [inline]

Definition at line 158 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::add_assign().

11.81.4.53 template<typename Boundary, typename Info> template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, Interval&>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::operator-= (const T & *x*) [inline]

Definition at line 164 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::sub_assign().

11.81.4.54 template<typename Boundary, typename Info> template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, Interval&>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::operator/= (const T & *x*) [inline]

Definition at line 176 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::div_assign().

11.81.4.55 template<typename Boundary, typename Info> template<typename T > Enable_If<Is_Singleton<T>::value || Is_Interval<T>::value, Interval&>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::operator= (const T & *x*) [inline]

Definition at line 151 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign().

11.81.4.56 template<typename Boundary, typename Info> Parma_Polyhedra_Library::Interval< Boundary, Info >::PPL_COMPILE_TIME_CHECK (!Info::store_special||!std::numeric_limits< Boundary >::has_infinity, "store_special is meaningless"" when boundary type may contains infinity") [private]

11.81.4.57 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::refine_existential (Relation_Symbol *rel*, const From & *x*) [inline]

Refines `to` according to the existential relation `rel` with `x`.

The `to` interval is restricted to become, upon successful exit, the smallest interval of its type that contains the set

$$\{\, a \in \texttt{to} \mid \exists b \in \texttt{x} \,.\, a \;\texttt{rel}\; b \,\}.$$

Returns

???

Definition at line 450 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Checked::eq, Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::f_is_singleton(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Checked::lt, and Parma_Polyhedra_Library::NOT_EQUAL.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::build().

11.81.4.58 template<typename To_Boundary , typename To_Info > template<typename From > Enable_If< Is_Singleton< From >::value||Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::refine_universal (Relation_Symbol *rel*, const From & *x*) [inline]

Refines `to` so that it satisfies the universal relation `rel` with `x`.

The `to` interval is restricted to become, upon successful exit, the smallest interval of its type that contains the set

$$\{\, a \in \texttt{to} \mid \forall b \in \texttt{x} : a \;\texttt{rel}\; b \,\}.$$

Returns

???

Definition at line 521 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Checked::eq, Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_Polyhedra_Library::f_is_singleton(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Interval< Boundary, Info

>::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_-Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_-Library::LESS_THAN, Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_-Polyhedra_Library::Checked::lt, Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::remove_inf(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::remove_sup(), Parma_Polyhedra_Library::Checked::Result, Parma_Polyhedra_Library::Interval_NS::SCALAR_INFO, and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.59 template<typename Boundary, typename Info> void Parma_-Polyhedra_Library::Interval< Boundary, Info >::remove_inf () [inline]

Definition at line 450 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), and Parma_Polyhedra_-Library::Interval< Boundary, Info >::is_empty().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

11.81.4.60 template<typename Boundary, typename Info> void Parma_-Polyhedra_Library::Interval< Boundary, Info >::remove_sup () [inline]

Definition at line 457 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), and Parma_Polyhedra_-Library::Interval< Boundary, Info >::is_empty().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal().

11.81.4.61 template<typename Boundary, typename Info> I_Result Parma_-Polyhedra_Library::Interval< Boundary, Info >::set_infinities () [inline]

Definition at line 419 of file Interval.defs.hh.

References Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Checked::Result, and Parma_-Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.62 template<typename Boundary, typename Info> template<typename From > Enable_-If<Is_Interval<From>::value, bool>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::simplify_using_context_assign (const From & *y*) [inline]

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`.

Returns

`false` if and only if the meet of `*this` and `y` is empty.

11.81.4.63 template<typename Boundary , typename Info > template<typename T > Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type Parma_Polyhedra_Library::Interval< Boundary, Info >::strictly_contains (const T & *y*) const [inline]

Definition at line 208 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::contains_-restriction(), Parma_Polyhedra_Library::eq_restriction(), Parma_Polyhedra_Library::Interval_NS::f_-info(), Parma_Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_-Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Checked::lt, Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

11.81.4.64 template<typename To_Boundary , typename To_Info > template<typename From1 , typename From2 > Enable_If<((Is_Singleton< From1 >::value||Is_Interval< From1 >::value)&&(Is_Singleton< From2 >::value||Is_Interval< From2 >::value)), I_Result >::type Parma_Polyhedra_Library::Interval< To_Boundary, To_Info >::sub_assign (const From1 & *x*, const From2 & *y*) [inline]

Definition at line 654 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::check_empty_arg(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Interval_NS::f_info(), Parma_-Polyhedra_Library::Interval_NS::f_lower(), Parma_Polyhedra_Library::f_OK(), Parma_Polyhedra_-Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::is_infinity(), Parma_Polyhedra_-Library::MINUS_INFINITY, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_-Library::Checked::Result, and Parma_Polyhedra_Library::sub_restriction().

Referenced by Parma_Polyhedra_Library::operator-(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::operator-=().

11.81.4.65 template<typename Boundary , typename Info > void Parma_Polyhedra_-Library::Interval< Boundary, Info >::swap (Interval< Boundary, Info > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 55 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::lower(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::swap().

11.81.4.66 template<typename Boundary, typename Info> void Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 441 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.67 template<typename Boundary , typename Info > memory_size_type Parma_Polyhedra_Library::Interval< Boundary, Info >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 49 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::external_memory_in_bytes().

11.81.4.68 template<typename Boundary, typename Info> const Boundary& Parma_Polyhedra_Library::Interval< Boundary, Info >::upper () const [inline]

Definition at line 204 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_.

11.81.4.69 template<typename Boundary, typename Info> Boundary& Parma_Polyhedra_Library::Interval< Boundary, Info >::upper () [inline]

Definition at line 200 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_dump(), Parma_Polyhedra_Library::Interval< Boundary, Info >::ascii_load(), Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::can_be_exactly_joined_to(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::difference_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::external_memory_in_bytes(), Parma_Polyhedra_Library::Interval_NS::f_upper(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_disjoint_from(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_infinity(), Parma_Polyhedra_Library::Interval< Boundary,

Info >::is_singleton(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::neg_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), Parma_Polyhedra_Library::Interval< Boundary, Info >::refine_universal(), Parma_Polyhedra_Library::Interval< Boundary, Info >::set_infinities(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::strictly_contains(), Parma_Polyhedra_Library::Interval< Boundary, Info >::swap(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_constraint(), Parma_-Polyhedra_Library::Interval< Boundary, Info >::upper_extend(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_domain_sup(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open(), Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize(), and Parma_Polyhedra_-Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.70 template<typename Boundary, typename Info> I_Constraint<boundary_type> Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_constraint () const [inline]

Definition at line 214 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open().

11.81.4.71 template<typename Boundary, typename Info> template<typename C > Enable_If<Is_Same_Or_Derived<I_Constraint_Base, C>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_extend (const C & *c*) [inline]

11.81.4.72 template<typename Boundary, typename Info> I_Result Parma_-Polyhedra_Library::Interval< Boundary, Info >::upper_extend () [inline]

Definition at line 302 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), and Parma_Polyhedra_-Library::Interval< Boundary, Info >::upper().

11.81.4.73 template<typename Boundary, typename Info> bool Parma_Polyhedra_-Library::Interval< Boundary, Info >::upper_is_boundary_infinity () const [inline]

Definition at line 266 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_-Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_bounded(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign().

11.81.4.74 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_domain_sup () const [inline]

Definition at line 276 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::is_universe().

11.81.4.75 template<typename Boundary, typename Info> bool Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_open () const [inline]

Definition at line 256 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::OK(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::contains_integer_point(), Parma_Polyhedra_Library::Interval< Boundary, Info >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_topologically_closed(), Parma_Polyhedra_Library::Interval< Boundary, Info >::topological_closure_assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_constraint().

11.81.4.76 template<typename Boundary, typename Info> Result Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize () const [inline, private]

Definition at line 120 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Checked::Result, Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::w_info().

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::normalize().

11.81.4.77 template<typename Boundary, typename Info> Info& Parma_Polyhedra_Library::Interval< Boundary, Info >::w_info () const [inline, private]

Definition at line 96 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_normalize(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_normalize().

11.81.4.78 template<typename Boundary, typename Info> template<typename From > Enable_If<Is_Singleton<From>::value || Is_Interval<From>::value, I_Result>::type Parma_Polyhedra_Library::Interval< Boundary, Info >::wrap_assign (Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, const From & *refinement*) [inline]

Definition at line 518 of file Interval.defs.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::info(), Parma_Polyhedra_Library::Interval< Boundary, Info >::intersect_assign(), Parma_Polyhedra_Library::Interval< Boundary, Info >::is_empty(), Parma_Polyhedra_Library::Interval< Boundary, Info >::join_assign(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Interval< Boundary, Info >::lower(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_extend(), Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_is_boundary_infinity(), Parma_Polyhedra_Library::Checked::Result, Parma_Polyhedra_Library::SIGNED_2_COMPLEMENT, Parma_Polyhedra_Library::UNSIGNED, Parma_Polyhedra_Library::Interval< Boundary, Info >::upper(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_is_boundary_infinity().

11.81.5 Friends And Related Function Documentation

11.81.5.1 template<typename Boundary , typename Info > void swap (Parma_Polyhedra_Library::Interval< Boundary, Info > & *x*, Parma_Polyhedra_Library::Interval< Boundary, Info > & *y*) [related]

Definition at line 1198 of file Interval.inlines.hh.

References Parma_Polyhedra_Library::Interval< Boundary, Info >::swap().

11.81.6 Member Data Documentation

11.81.6.1 template<typename Boundary, typename Info> Boundary Parma_Polyhedra_Library::Interval< Boundary, Info >::lower_ [private]

Definition at line 838 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::lower().

11.81.6.2 template<typename Boundary, typename Info> Boundary Parma_Polyhedra_Library::Interval< Boundary, Info >::upper_ [private]

Definition at line 839 of file Interval.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval< Boundary, Info >::assign(), and Parma_Polyhedra_Library::Interval< Boundary, Info >::upper().

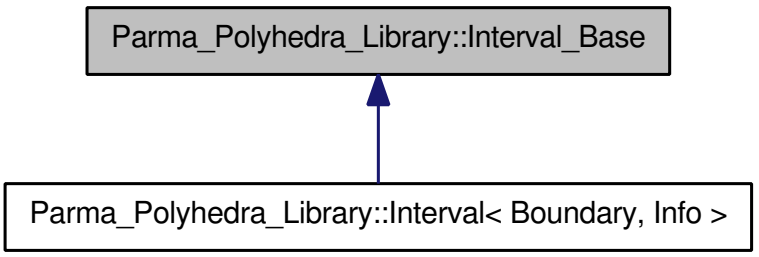
The documentation for this class was generated from the following files:

- Interval.defs.hh
- Interval.inlines.hh

11.82 Parma_Polyhedra_Library::Interval_Base Struct Reference

`#include <Interval.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Base:



11.82.1 Detailed Description

Definition at line 47 of file Interval.defs.hh.

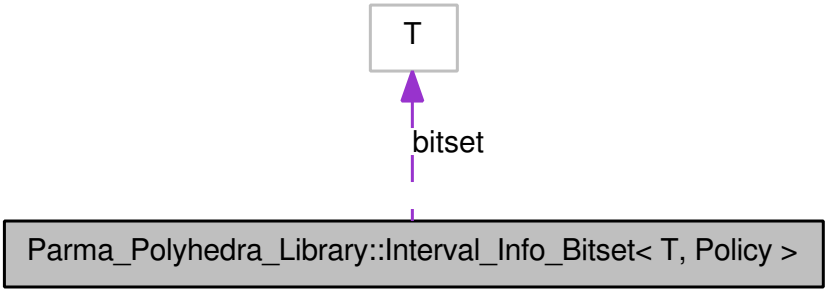
The documentation for this struct was generated from the following file:

- Interval.defs.hh

11.83 Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > Class Template Reference

`#include <Interval_Info.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >:



Public Member Functions

- const_bool_nodef (may_be_empty, Policy::may_be_empty)
- const_bool_nodef (may_contain_infinity, Policy::may_contain_infinity)
- const_bool_nodef (check_inexact, Policy::check_inexact)
- const_bool_nodef (store_special, Policy::store_special)
- const_bool_nodef (store_open, Policy::store_open)
- const_bool_nodef (cache_normalized, Policy::cache_normalized)
- const_bool_nodef (cache_empty, Policy::cache_empty)
- const_bool_nodef (cache_singleton, Policy::cache_singleton)
- const_int_nodef (lower_special_bit, Policy::next_bit)
- const_int_nodef (lower_open_bit, lower_special_bit+store_special)
- const_int_nodef (lower_normalized_bit, lower_open_bit+store_open)
- const_int_nodef (upper_special_bit, lower_normalized_bit+cache_normalized)
- const_int_nodef (upper_open_bit, upper_special_bit+store_special)
- const_int_nodef (upper_normalized_bit, upper_open_bit+store_open)
- const_int_nodef (cardinality_is_bit, upper_normalized_bit+cache_normalized)
- const_int_nodef (cardinality_0_bit, cardinality_is_bit+(cache_empty||cache_singleton))
- const_int_nodef (cardinality_1_bit, cardinality_0_bit+cache_empty)
- const_int_nodef (next_bit, cardinality_1_bit+cache_singleton)
- Interval_Info_Bitset ()
- void clear ()
- void clear_boundary_properties (Boundary_Type t)
- void set_boundary_property (Boundary_Type t, const Boundary_NS::Property &p, bool value=true)
- bool get_boundary_property (Boundary_Type t, const Boundary_NS::Property &p) const
- void set_interval_property (const Interval_NS::Property &p, bool value=true)
- bool get_interval_property (Interval_NS::Property p) const
- void swap (Interval_Info_Bitset &y)

  Swaps `*this` *with* `y`.

- void ascii_dump (std::ostream &s) const
- bool ascii_load (std::istream &s)

Protected Attributes

- T bitset

Related Functions

(Note that these are not member functions.)

- template<typename T , typename Policy >
  void swap (Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > &x, Parma_Polyhedra_-Library::Interval_Info_Bitset< T, Policy > &y)

11.83.1 Detailed Description

template<typename T, typename Policy> class Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >

Definition at line 164 of file Interval_Info.defs.hh.

11.83.2 Constructor & Destructor Documentation

11.83.2.1 template<typename T, typename Policy> Parma_Polyhedra_-Library::Interval_Info_Bitset< T, Policy >::Interval_Info_Bitset () [inline]

Definition at line 184 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::clear().

11.83.3 Member Function Documentation

11.83.3.1 template<typename T , typename Policy > void Parma_Polyhedra_-Library::Interval_Info_Bitset< T, Policy >::ascii_dump (std::ostream & *s*) const [inline]

Definition at line 78 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset.

11.83.3.2 template<typename T , typename Policy > bool Parma_Polyhedra_-Library::Interval_Info_Bitset< T, Policy >::ascii_load (std::istream & *s*) [inline]

Definition at line 86 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset.

11.83.3.3 template<typename T, typename Policy> void Parma_Polyhedra_Library::Interval_-Info_Bitset< T, Policy >::clear () [inline]

Definition at line 190 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset, and Parma_Polyhedra_Library::Interval_NS::reset_bits().

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::Interval_Info_Bitset().

11.83.3.4 template<typename T, typename Policy> void Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::clear_boundary_properties (Boundary_Type *t*) [inline]

Definition at line 193 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_boundary_property().

11.83.3.5 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (cache_singleton, Policy::cache_singleton)

11.83.3.6 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (cache_empty, Policy::cache_empty)

11.83.3.7 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (cache_normalized, Policy::cache_normalized)

11.83.3.8 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (store_open, Policy::store_open)

11.83.3.9 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (store_special, Policy::store_special)

11.83.3.10 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (check_inexact, Policy::check_inexact)

11.83.3.11 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (may_contain_infinity, Policy::may_contain_infinity)

11.83.3.12 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_bool_nodef (may_be_empty, Policy::may_be_empty)

11.83.3.13 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (next_bit, cardinality_1_bit+ *cache_singleton*)

11.83.3.14 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (cardinality_1_bit, cardinality_0_bit+ *cache_empty*)

11.83.3.15 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (cardinality_0_bit, cardinality_is_bit+cache_empty||cache_singleton)

11.83.3.16 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (cardinality_is_bit, upper_normalized_bit+ *cache_normalized*)

11.83.3.17 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (upper_normalized_bit, upper_open_bit+ *store_open*)

11.83.3.18 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (upper_open_bit, upper_special_bit+ *store_special*)

11.83.3.19 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (upper_special_bit, lower_normalized_bit+ *cache_normalized*)

11.83.3.20 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (lower_normalized_bit, lower_open_bit+ *store_open*)

11.83.3.21 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (lower_open_bit, lower_special_bit+ *store_special*)

11.83.3.22 template<typename T, typename Policy> Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::const_int_nodef (lower_special_bit, Policy::next_bit)

11.83.3.23 template<typename T, typename Policy> bool Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_boundary_property (Boundary_Type *t*, const Boundary_NS::Property & *p*) const [inline]

Definition at line 227 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset, and Parma_Polyhedra_Library::Interval_NS::get_bit().

11.83.3.24 template<typename T, typename Policy> bool Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_interval_property (Interval_NS::Property *p*) const [inline]

Definition at line 272 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset, Parma_Polyhedra_Library::Interval_NS::Property::CARDINALITY_0_, Parma_Polyhedra_-

Library::Interval_NS::Property::CARDINALITY_1_, Parma_Polyhedra_Library::Interval_-NS::Property::CARDINALITY_IS_, Parma_Polyhedra_Library::Interval_NS::get_bit(), and Parma_-Polyhedra_Library::Interval_NS::Property::type.

11.83.3.25 template<typename T, typename Policy> void Parma_Polyhedra_Library::Interval_-Info_Bitset< T, Policy >::set_boundary_property (Boundary_Type *t*, const Boundary_NS::Property & *p*, bool *value* = `true`) `[inline]`

Definition at line 197 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset, and Parma_Polyhedra_-Library::Interval_NS::set_bit().

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::clear_boundary_-properties().

11.83.3.26 template<typename T, typename Policy> void Parma_Polyhedra_Library::Interval_-Info_Bitset< T, Policy >::set_interval_property (const Interval_NS::Property & *p*, bool *value* = `true`) `[inline]`

Definition at line 254 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset, Parma_-Polyhedra_Library::Interval_NS::Property::CARDINALITY_0_, Parma_Polyhedra_-Library::Interval_NS::Property::CARDINALITY_1_, Parma_Polyhedra_Library::Interval_-NS::Property::CARDINALITY_IS_, Parma_Polyhedra_Library::Interval_NS::set_bit(), and Parma_-Polyhedra_Library::Interval_NS::Property::type.

11.83.3.27 template<typename T , typename Policy > void Parma_Polyhedra_Library::Interval_-Info_Bitset< T, Policy >::swap (Interval_Info_Bitset< T, Policy > & *y*) `[inline]`

Swaps `*this` with `y`.

Definition at line 72 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset.

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::swap().

11.83.4 Friends And Related Function Documentation

11.83.4.1 template<typename T , typename Policy > void swap (Parma_-Polyhedra_Library::Interval_Info_Bitset< T, Policy > & *x*, Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy > & *y*) `[related]`

Definition at line 115 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::swap().

11.83.5 Member Data Documentation

11.83.5.1 template<typename T, typename Policy> T Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::bitset [protected]

Definition at line 293 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::ascii_dump(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::ascii_load(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::clear(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_boundary_property(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_interval_property(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_boundary_property(), Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_interval_property(), and Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::swap().

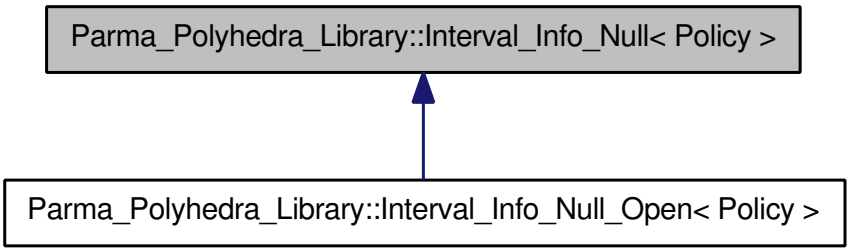
The documentation for this class was generated from the following files:

- Interval_Info.defs.hh
- Interval_Info.inlines.hh

11.84 Parma_Polyhedra_Library::Interval_Info_Null< Policy > Class Template Reference

`#include <Interval_Info.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Info_Null< Policy >:



Public Member Functions

- const_bool_nodef (may_be_empty, Policy::may_be_empty)
- const_bool_nodef (may_contain_infinity, Policy::may_contain_infinity)
- const_bool_nodef (check_inexact, Policy::check_inexact)
- const_bool_nodef (store_special, false)
- const_bool_nodef (store_open, false)
- const_bool_nodef (cache_normalized, false)
- const_bool_nodef (cache_empty, false)
- const_bool_nodef (cache_singleton, false)
- void clear ()
- void clear_boundary_properties (Boundary_Type)
- template<typename Property >
  void set_boundary_property (Boundary_Type, const Property &, typename Property::Value=Property::default_value)

- template<typename Property >
  Property::Value get_boundary_property (Boundary_Type, const Property &) const
- template<typename Property >
  void set_interval_property (const Property &, typename Property::Value=Property::default_value)
- template<typename Property >
  Property::Value get_interval_property (const Property &) const
- void swap (Interval_Info_Null &y)

  Swaps `*this` *with* `y`.

- void ascii_dump (std::ostream &s) const
- bool ascii_load (std::istream &s)

Related Functions

(Note that these are not member functions.)

- template<typename Policy >
  void swap (Parma_Polyhedra_Library::Interval_Info_Null< Policy > &x, Parma_Polyhedra_Library::Interval_Info_Null< Policy > &y)

11.84.1 Detailed Description

template<typename Policy> class Parma_Polyhedra_Library::Interval_Info_Null< Policy >

Definition at line 102 of file Interval_Info.defs.hh.

11.84.2 Member Function Documentation

11.84.2.1 template<typename Policy > void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::ascii_dump (std::ostream & *s*) const [inline]

Reimplemented in Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >.

Definition at line 38 of file Interval_Info.inlines.hh.

11.84.2.2 template<typename Policy > bool Parma_Polyhedra_Library::Interval_Info_Null< Policy >::ascii_load (std::istream & *s*) [inline]

Reimplemented in Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >.

Definition at line 43 of file Interval_Info.inlines.hh.

11.84.2.3 template<typename Policy> void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::clear () [inline]

Definition at line 112 of file Interval_Info.defs.hh.

11.84.2.4 template<typename Policy> void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::clear_boundary_properties (Boundary_Type) [inline]

Definition at line 114 of file Interval_Info.defs.hh.

11.84.2.5 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (cache_singleton, false)

11.84.2.6 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (cache_empty, false)

11.84.2.7 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (cache_normalized, false)

11.84.2.8 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (store_open, false)

11.84.2.9 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (store_special, false)

11.84.2.10 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (check_inexact, Policy::check_inexact)

11.84.2.11 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (may_contain_infinity, Policy::may_contain_infinity)

11.84.2.12 template<typename Policy> Parma_Polyhedra_Library::Interval_Info_Null< Policy >::const_bool_nodef (may_be_empty, Policy::may_be_empty)

11.84.2.13 template<typename Policy> template<typename Property > Property::Value Parma_Polyhedra_Library::Interval_Info_Null< Policy >::get_boundary_property (Boundary_Type, const Property &) const [inline]

Definition at line 121 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_NS::Property::unsupported_value.

11.84.2.14 template<typename Policy> template<typename Property > Property::Value Parma_Polyhedra_Library::Interval_Info_Null< Policy >::get_interval_property (const Property &) const [inline]

Definition at line 128 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_NS::Property::unsupported_value.

11.84.2.15 template<typename Policy> template<typename Property > void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::set_boundary_property (Boundary_Type, const Property &, typename Property::Value = Property::default_value) [inline]

Definition at line 118 of file Interval_Info.defs.hh.

11.84.2.16 template<typename Policy> template<typename Property > void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::set_interval_property (const Property &, typename Property::Value = Property::default_value) [inline]

Definition at line 125 of file Interval_Info.defs.hh.

11.84.2.17 template<typename Policy > void Parma_Polyhedra_Library::Interval_Info_Null< Policy >::swap (Interval_Info_Null< Policy > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 33 of file Interval_Info.inlines.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Null< Policy >::swap().

11.84.3 Friends And Related Function Documentation

11.84.3.1 template<typename Policy > void swap (Parma_Polyhedra_Library::Interval_Info_Null< Policy > & *x*, Parma_Polyhedra_Library::Interval_Info_Null< Policy > & *y*) [related]

Definition at line 105 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Null< Policy >::swap().

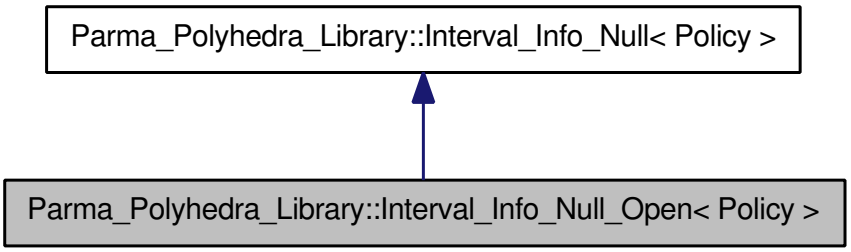
The documentation for this class was generated from the following files:

- Interval_Info.defs.hh
- Interval_Info.inlines.hh

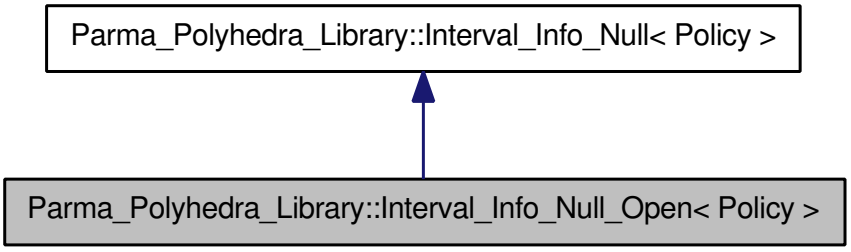
11.85 Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy > Class Template Reference

`#include <Interval_Info.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >:



Collaboration diagram for Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >:



Public Member Functions

- const_bool_nodef (store_open, true)
- Interval_Info_Null_Open (bool o)
- bool get_boundary_property (Boundary_Type, const Boundary_NS::Property &p) const
- void ascii_dump (std::ostream &s) const
- bool ascii_load (std::istream &s)

Private Attributes

- bool open

11.85.1 Detailed Description

template<typename Policy> class Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >

Definition at line 140 of file Interval_Info.defs.hh.

11.85.2 Constructor & Destructor Documentation

11.85.2.1 template<typename Policy > Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::Interval_Info_Null_Open (bool *o*) [inline]

Definition at line 143 of file Interval_Info.defs.hh.

11.85.3 Member Function Documentation

11.85.3.1 template<typename Policy > void Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::ascii_dump (std::ostream & *s*) const [inline]

Reimplemented from Parma_Polyhedra_Library::Interval_Info_Null< Policy >.

Definition at line 49 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::open.

11.85.3.2 template<typename Policy > bool Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::ascii_load (std::istream & *s*) [inline]

Reimplemented from Parma_Polyhedra_Library::Interval_Info_Null< Policy >.

Definition at line 55 of file Interval_Info.inlines.hh.

References Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::open.

11.85.3.3 template<typename Policy > Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::const_bool_nodef (store_open, true)

11.85.3.4 template<typename Policy > bool Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::get_boundary_property (Boundary_Type, const Boundary_NS::Property & *p*) const [inline]

Definition at line 146 of file Interval_Info.defs.hh.

References Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::open.

11.85.4 Member Data Documentation

11.85.4.1 template<typename Policy > bool Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::open [private]

Definition at line 159 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::ascii_dump(), Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::ascii_load(), and Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >::get_boundary_property().

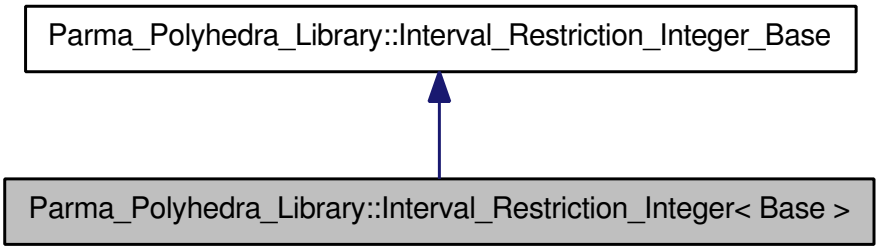
The documentation for this class was generated from the following files:

- Interval_Info.defs.hh
- Interval_Info.inlines.hh

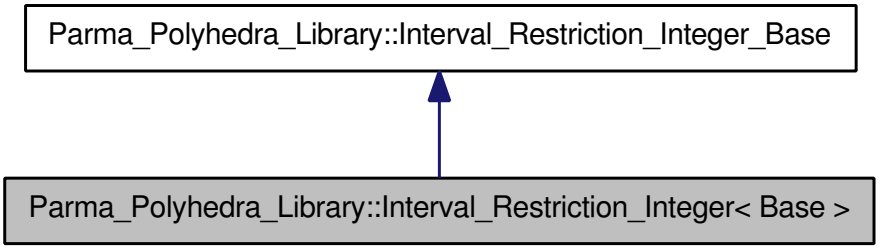
11.86 Parma_Polyhedra_Library::Interval_Restriction_Integer< Base > Class Template Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >:



Collaboration diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >:



Public Member Functions

- Interval_Restriction_Integer ()
- void set_integer (bool v=true)
- bool get_integer () const
- const_int_nodef (integer_bit, Base::next_bit)

- const_int_nodef (next_bit, integer_bit+1)
- bool has_restriction () const
- void normalize () const
- template<typename T >
  Result restrict (Rounding_Dir rdir, T &x, Result dir) const

11.86.1 Detailed Description

template<typename Base> class Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >

Definition at line 146 of file Interval_Restriction.defs.hh.

11.86.2 Constructor & Destructor Documentation

11.86.2.1 template<typename Base> Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::Interval_Restriction_Integer () [inline]

Definition at line 148 of file Interval_Restriction.defs.hh.

11.86.3 Member Function Documentation

11.86.3.1 template<typename Base> Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::const_int_nodef (next_bit, integer_bit+ *1*)

11.86.3.2 template<typename Base> Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::const_int_nodef (integer_bit, Base::next_bit)

11.86.3.3 template<typename Base> bool Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::get_integer () const [inline]

Definition at line 153 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_NS::get_bit().

Referenced by Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::has_restriction(), and Parma_Polyhedra_Library::output_restriction().

11.86.3.4 template<typename Base> bool Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::has_restriction () const [inline]

Definition at line 159 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::get_integer().

Referenced by Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::restrict().

11.86.3.5 template<typename Base> void Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::normalize () const [inline]

Definition at line 162 of file Interval_Restriction.defs.hh.

11.86.3.6 template<typename Base> template<typename T > Result Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::restrict (Rounding_Dir *rdir*, T & *x*, Result *dir*) const [inline]

Definition at line 165 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::has_restriction(), and Parma_Polyhedra_Library::is_integer().

11.86.3.7 template<typename Base> void Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >::set_integer (bool *v* = true) [inline]

Definition at line 150 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_NS::set_bit().

Referenced by Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::assign_restriction(), Parma_Polyhedra_Library::diff_restriction(), Parma_Polyhedra_Library::div_restriction(), Parma_Polyhedra_Library::intersect_restriction(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::neg_restriction(), and Parma_Polyhedra_Library::sub_restriction().

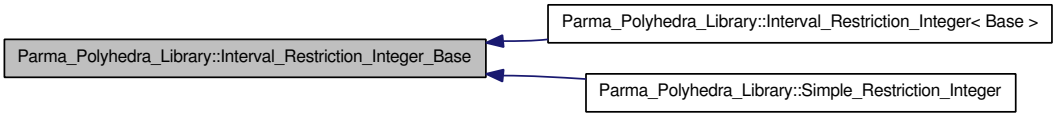
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

11.87 Parma_Polyhedra_Library::Interval_Restriction_Integer_Base Class Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer_Base:



11.87.1 Detailed Description

Definition at line 142 of file Interval_Restriction.defs.hh.

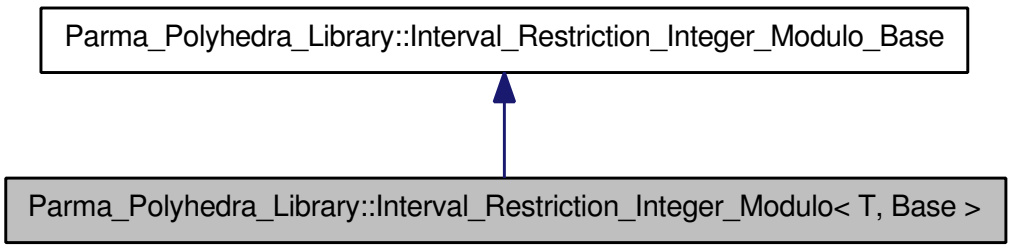
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

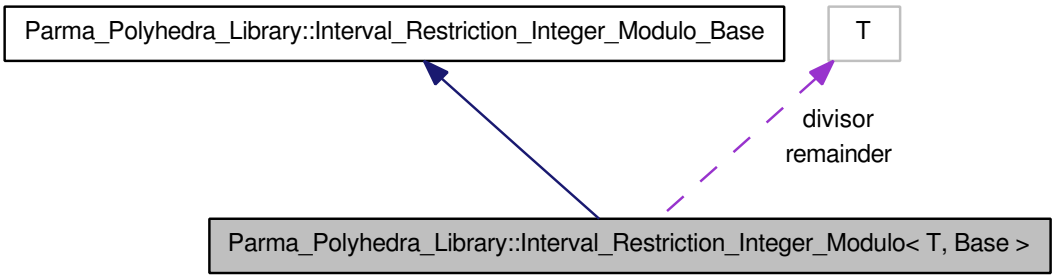
11.88 Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base > Class Template Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >:



Collaboration diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >:



Public Types

- typedef T modulo_type

Public Member Functions

- PPL_COMPILE_TIME_CHECK (std::numeric_limits< T >::is_exact,"type for modulo values must be exact")
- Interval_Restriction_Integer_Modulo ()
- bool has_restriction () const
- void clear ()
- void normalize () const
- template<typename V >
  Result restrict (Rounding_Dir rdir, V &x, Result dir) const

- void assign_or_swap (Interval_Restriction_Integer_Modulo &x)

Public Attributes

- T remainder
- T divisor

11.88.1 Detailed Description

template<typename T, typename Base> class Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >

Definition at line 319 of file Interval_Restriction.defs.hh.

11.88.2 Member Typedef Documentation

11.88.2.1 template<typename T, typename Base> typedef T Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::modulo_type

Definition at line 397 of file Interval_Restriction.defs.hh.

11.88.3 Constructor & Destructor Documentation

11.88.3.1 template<typename T, typename Base> Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::Interval_Restriction_Integer_Modulo () [inline]

Definition at line 323 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::clear().

11.88.4 Member Function Documentation

11.88.4.1 template<typename T, typename Base> void Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::assign_or_swap (Interval_Restriction_Integer_Modulo< T, Base > & *x*) [inline]

Definition at line 393 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

11.88.4.2 template<typename T, typename Base> void Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::clear () [inline]

Definition at line 331 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, and Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder.

Referenced by Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::Interval_Restriction_Integer_Modulo().

11.88.4.3 template<typename T, typename Base> bool Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::has_restriction () const [inline]

Definition at line 328 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor.

Referenced by Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::restrict().

11.88.4.4 template<typename T, typename Base> void Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::normalize () const [inline]

Definition at line 336 of file Interval_Restriction.defs.hh.

11.88.4.5 template<typename T, typename Base> Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::PPL_COMPILE_TIME_CHECK (std::numeric_limits< T >::is_exact, "type for modulo values must be exact")

11.88.4.6 template<typename T, typename Base> template<typename V > Result Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::restrict (Rounding_Dir *rdir*, V & *x*, Result *dir*) const [inline]

Definition at line 339 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor, Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::has_restriction(), and Parma_Polyhedra_Library::Checked::Result.

11.88.5 Member Data Documentation

11.88.5.1 template<typename T, typename Base> T Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::divisor

Definition at line 399 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::assign_or_swap(), Parma_Polyhedra_Library::assign_restriction(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::clear(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::has_restriction(), Parma_Polyhedra_Library::intersect_restriction(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::output_restriction(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::restrict(), Parma_Polyhedra_Library::set_integer(), Parma_Polyhedra_Library::set_unrestricted(), and Parma_Polyhedra_Library::sub_restriction().

11.88.5.2 template<typename T, typename Base> T Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::remainder

Definition at line 398 of file Interval_Restriction.defs.hh.

Referenced by Parma_Polyhedra_Library::add_restriction(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::assign_or_swap(), Parma_Polyhedra_Library::assign_restriction(), Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >::clear(), Parma_Polyhedra_Library::intersect_restriction(), Parma_Polyhedra_Library::join_restriction(), Parma_Polyhedra_Library::mul_restriction(), Parma_Polyhedra_Library::output_restriction(), Parma_Polyhedra_Library::set_integer(), Parma_Polyhedra_Library::set_unrestricted(), and Parma_Polyhedra_Library::sub_restriction().

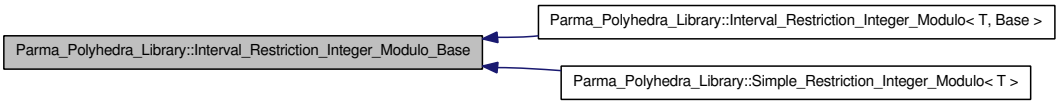
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

11.89 Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base Class Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base:



11.89.1 Detailed Description

Definition at line 315 of file Interval_Restriction.defs.hh.

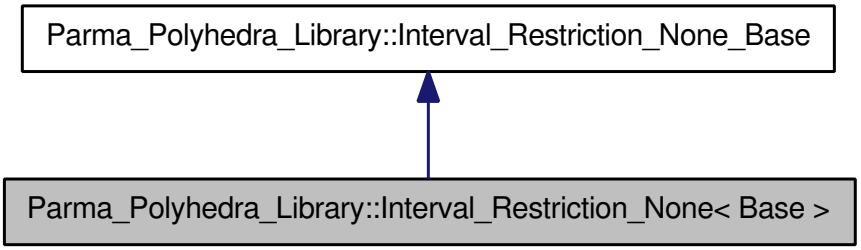
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

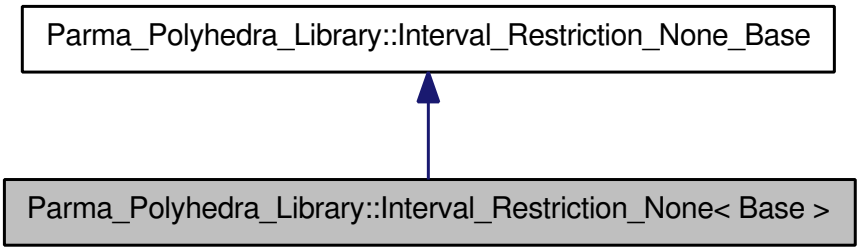
11.90 Parma_Polyhedra_Library::Interval_Restriction_None< Base > Class Template Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_None< Base >:



Collaboration diagram for Parma_Polyhedra_Library::Interval_Restriction_None< Base >:



Public Member Functions

- Interval_Restriction_None ()
- template<typename T >
  Interval_Restriction_None (const T &init)

11.90.1 Detailed Description

template<typename Base> class Parma_Polyhedra_Library::Interval_Restriction_None< Base >

Definition at line 131 of file Interval_Restriction.defs.hh.

11.90.2 Constructor & Destructor Documentation

11.90.2.1 template<typename Base > Parma_Polyhedra_Library::Interval_Restriction_None< Base >::Interval_Restriction_None () [inline]

Definition at line 134 of file Interval_Restriction.defs.hh.

11.90.2.2 template<typename Base > template<typename T > Parma_Polyhedra_Library::Interval_Restriction_None< Base >::Interval_Restriction_None (const T & *init*) [inline]

Definition at line 137 of file Interval_Restriction.defs.hh.

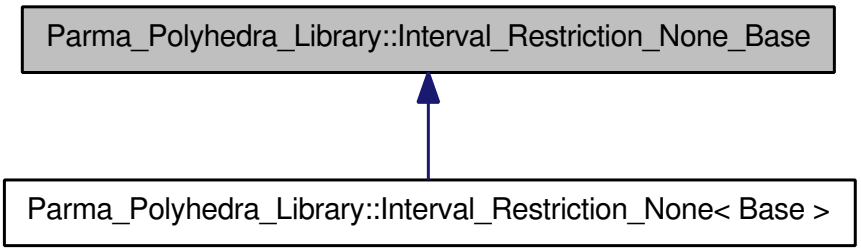
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

11.91 Parma_Polyhedra_Library::Interval_Restriction_None_Base Class Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Interval_Restriction_None_Base:



Public Member Functions

- bool has_restriction () const
- void normalize () const
- template<typename T >
  Result restrict (Rounding_Dir, T &, Result dir) const

11.91.1 Detailed Description

Definition at line 48 of file Interval_Restriction.defs.hh.

11.91.2 Member Function Documentation

11.91.2.1 bool Parma_Polyhedra_Library::Interval_Restriction_None_Base::has_restriction () const [inline]

Definition at line 50 of file Interval_Restriction.defs.hh.

11.91.2.2 void Parma_Polyhedra_Library::Interval_Restriction_None_Base::normalize () const [inline]

Definition at line 53 of file Interval_Restriction.defs.hh.

11.91.2.3 template<typename T > Result Parma_Polyhedra_Library::Interval_Restriction_None_Base::restrict (Rounding_Dir, T &, Result *dir*) const [inline]

Definition at line 56 of file Interval_Restriction.defs.hh.

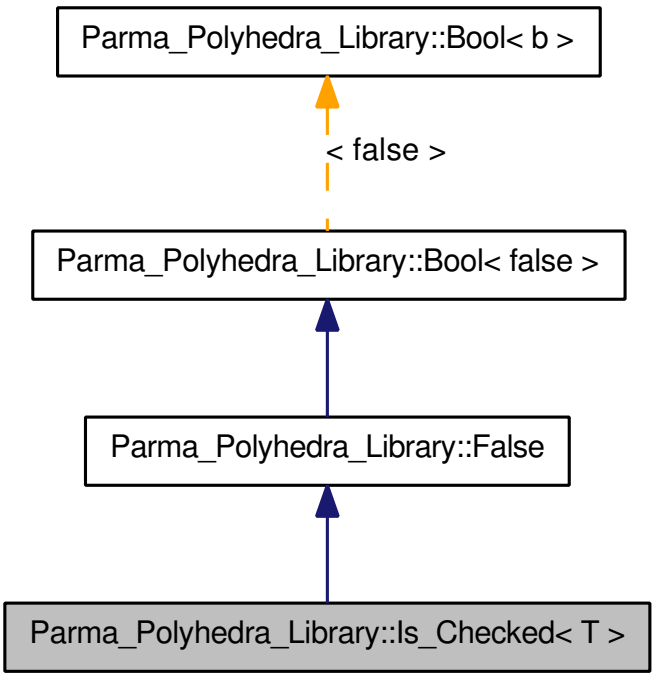
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

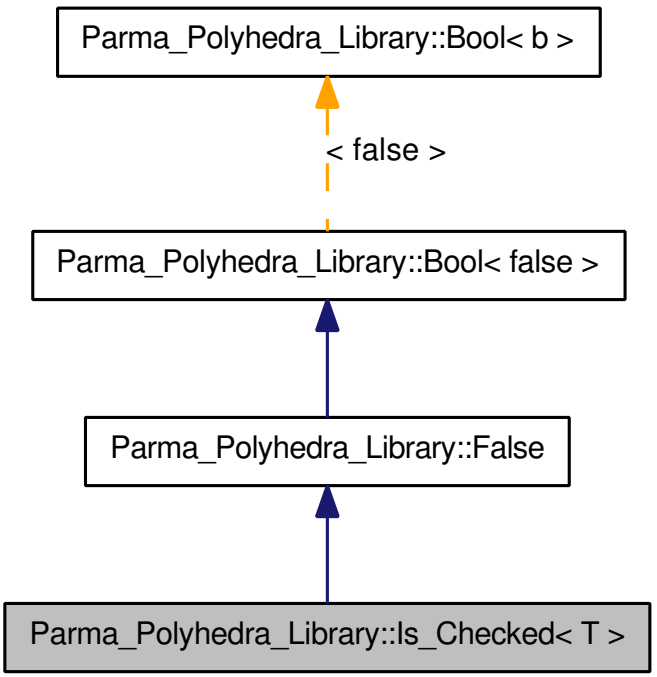
11.92 Parma_Polyhedra_Library::Is_Checked< T > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Checked< T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Checked< T >:



11.92.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Checked< T >

Definition at line 139 of file Checked_Number.defs.hh.

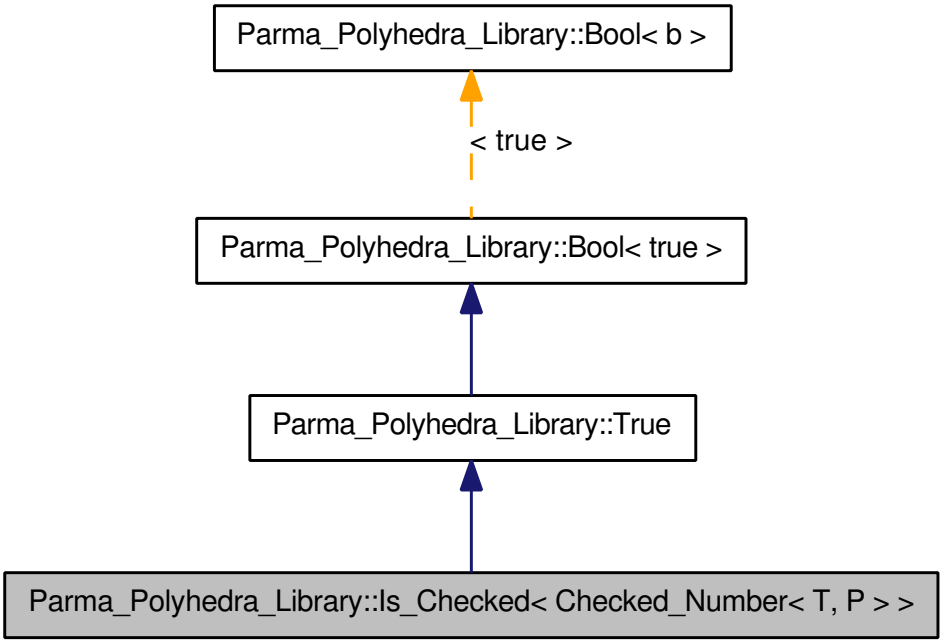
The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

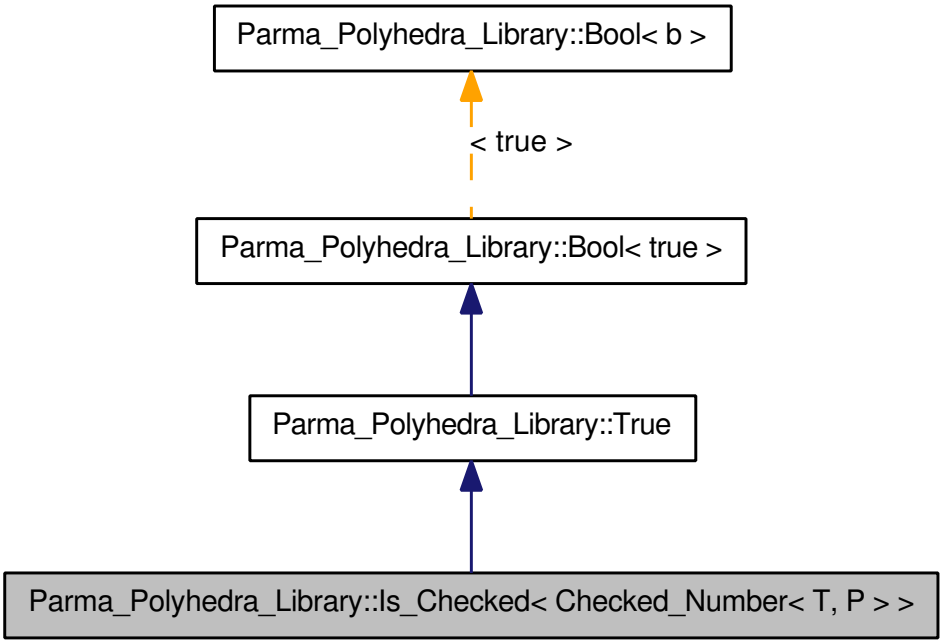
11.93 Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >:



11.93.1 Detailed Description

template<typename T, typename P> struct Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >

Definition at line 143 of file Checked_Number.defs.hh.

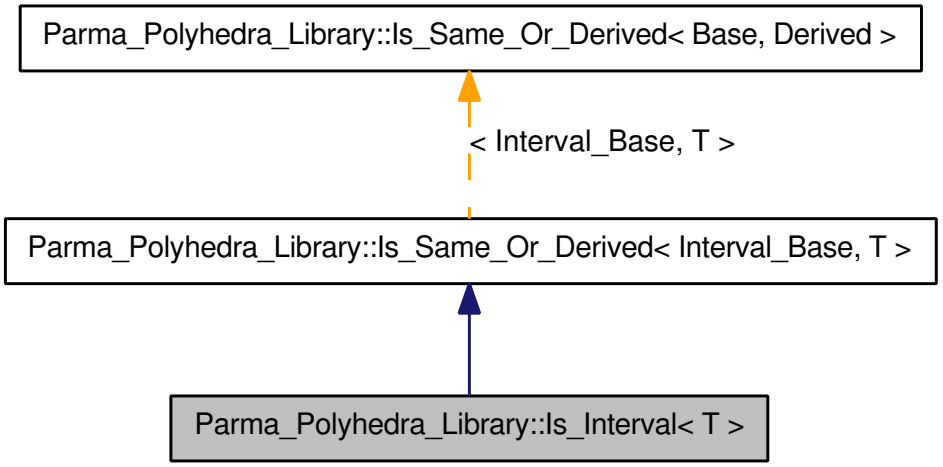
The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

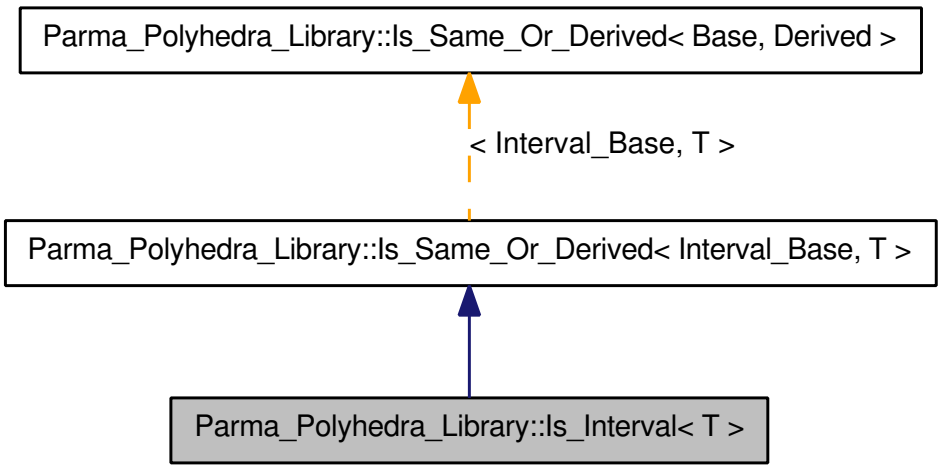
11.94 Parma_Polyhedra_Library::Is_Interval< T > Struct Template Reference

`#include <Interval.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Interval< T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Interval< T >:



11.94.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Interval< T >

Definition at line 57 of file Interval.defs.hh.

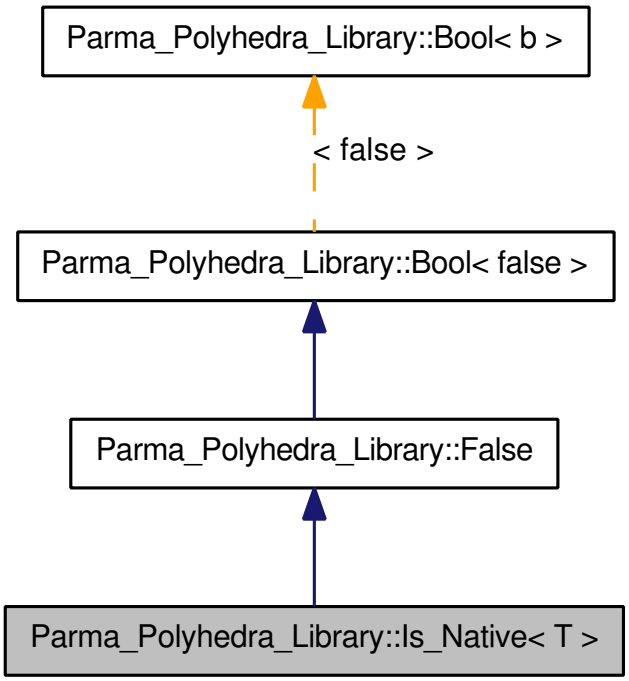
The documentation for this struct was generated from the following file:

- Interval.defs.hh

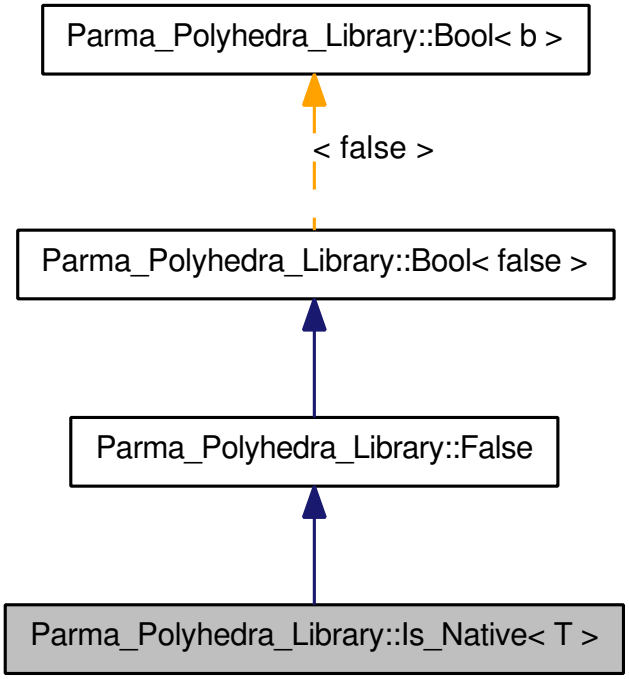
11.95 Parma_Polyhedra_Library::Is_Native< T > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< T >:



11.95.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Native< T >

Definition at line 311 of file meta_programming.hh.

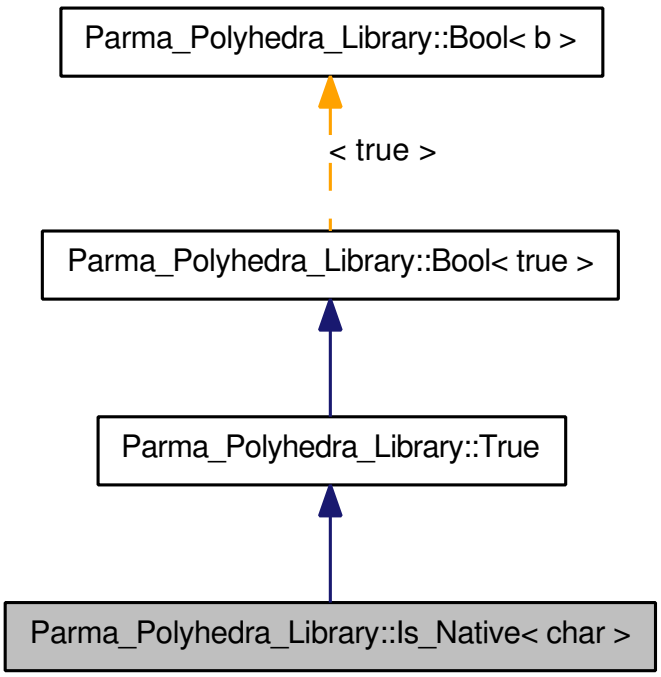
The documentation for this struct was generated from the following file:

- meta_programming.hh

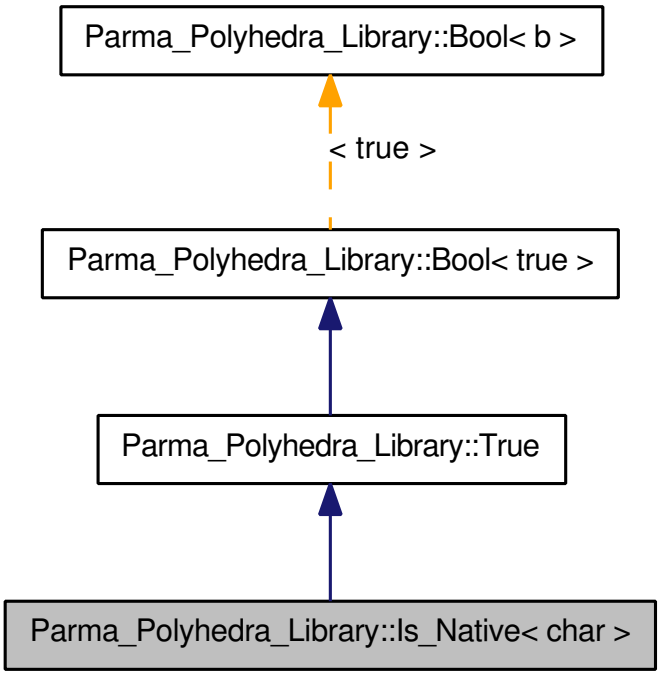
11.96 Parma_Polyhedra_Library::Is_Native< char > Struct Template Reference

```
#include <meta_programming.hh>
```

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< char >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< char >:



11.96.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< char >

Definition at line 314 of file meta_programming.hh.

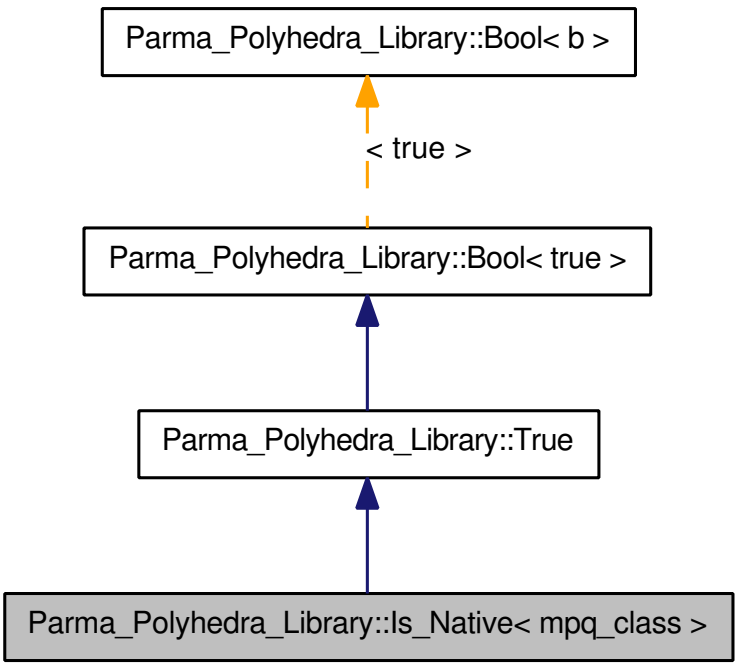
The documentation for this struct was generated from the following file:

- meta_programming.hh

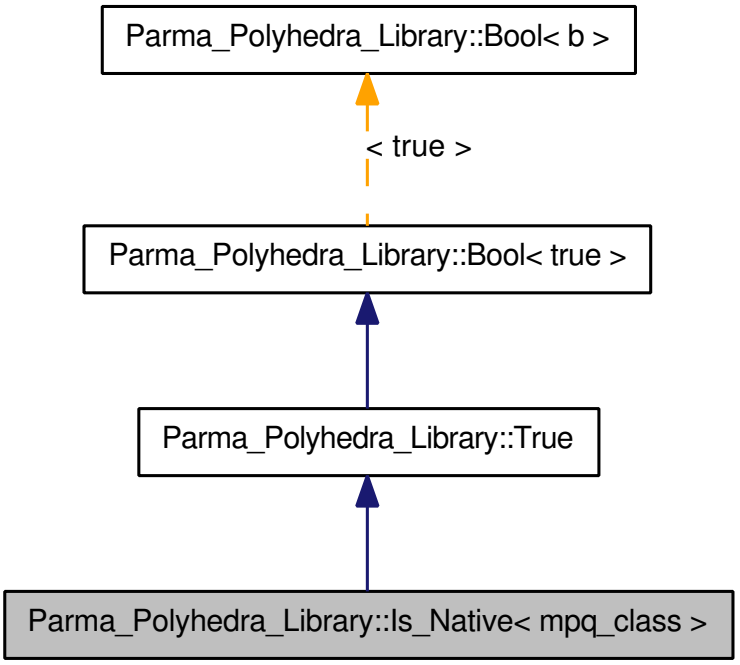
11.97 Parma_Polyhedra_Library::Is_Native< mpq_class > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< mpq_class >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< mpq_class >:



11.97.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< mpq_class >

Definition at line 338 of file meta_programming.hh.

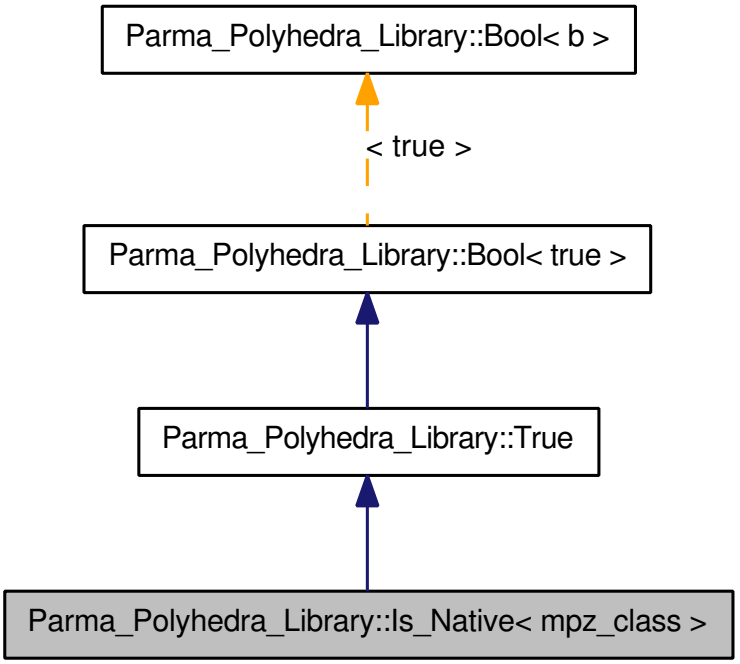
The documentation for this struct was generated from the following file:

- meta_programming.hh

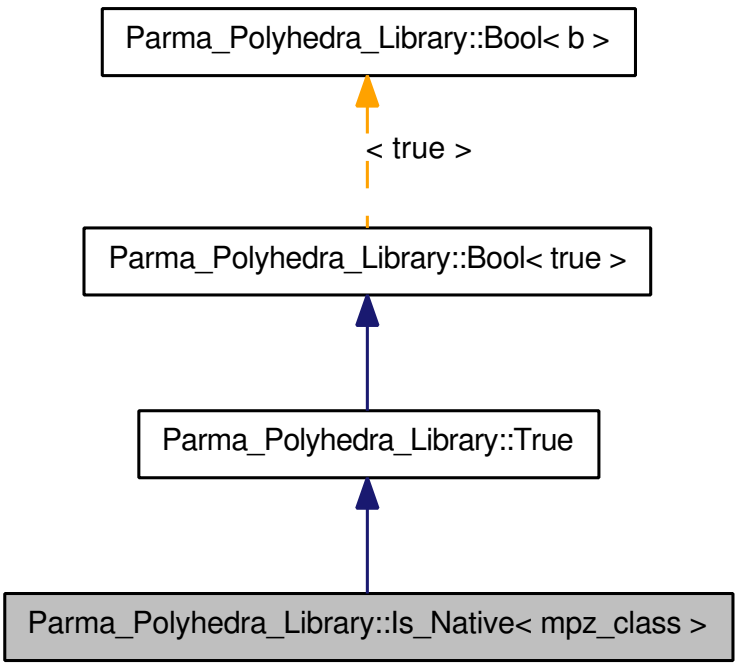
11.98 Parma_Polyhedra_Library::Is_Native< mpz_class > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< mpz_class >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< mpz_class >:



11.98.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< mpz_class >

Definition at line 336 of file meta_programming.hh.

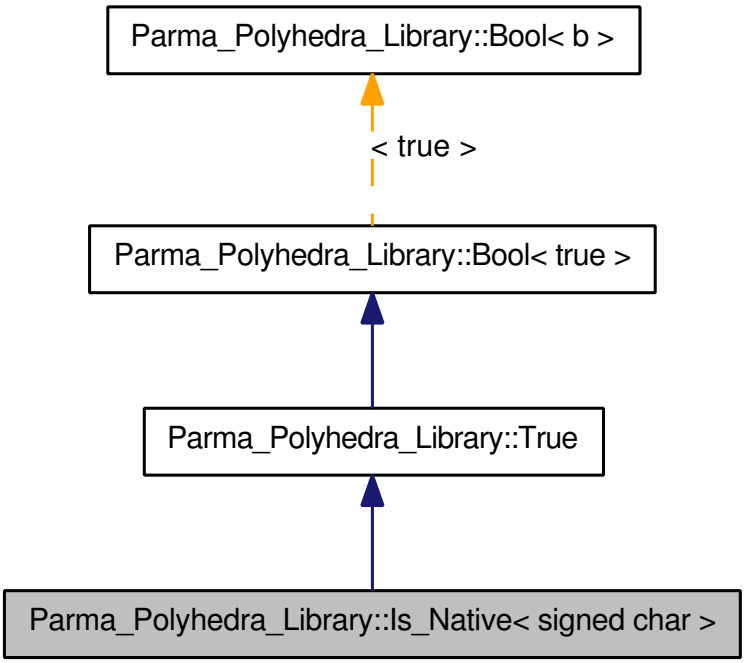
The documentation for this struct was generated from the following file:

- meta_programming.hh

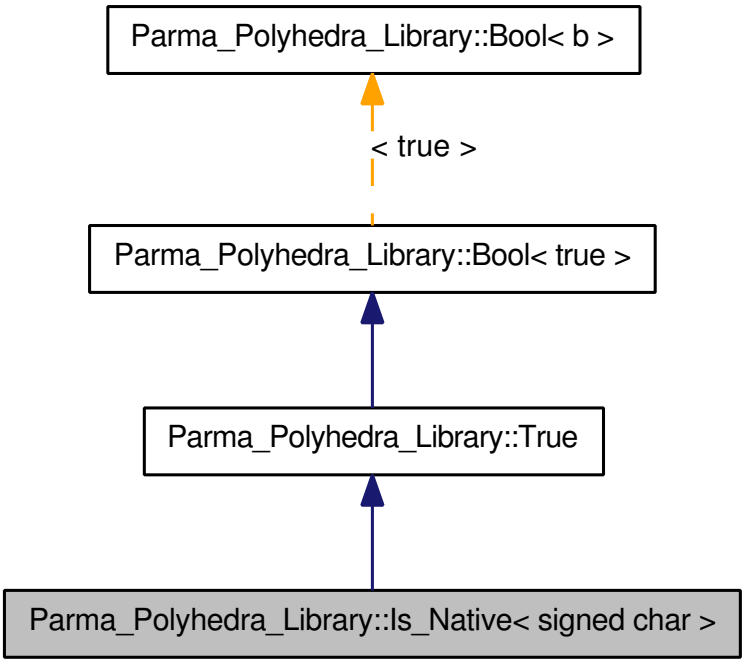
11.99 Parma_Polyhedra_Library::Is_Native< signed char > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< signed char >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< signed char >:



11.99.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< signed char >

Definition at line 315 of file meta_programming.hh.

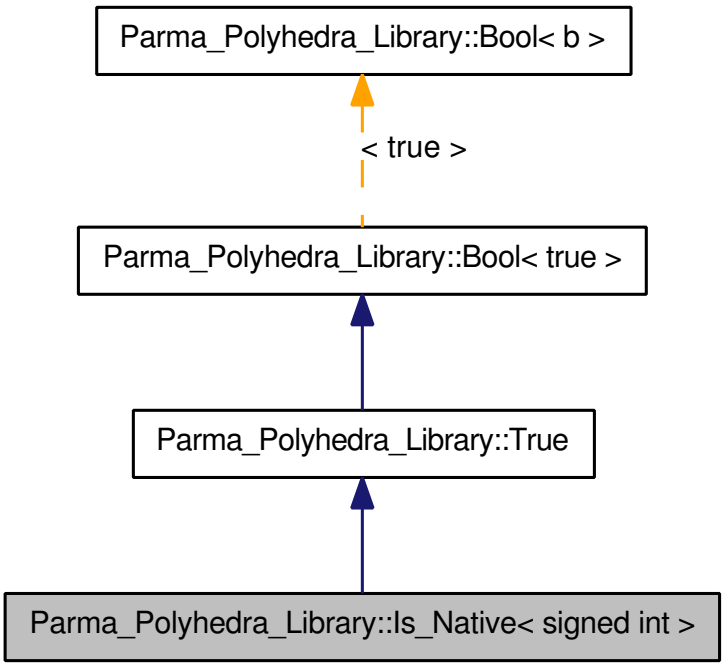
The documentation for this struct was generated from the following file:

- meta_programming.hh

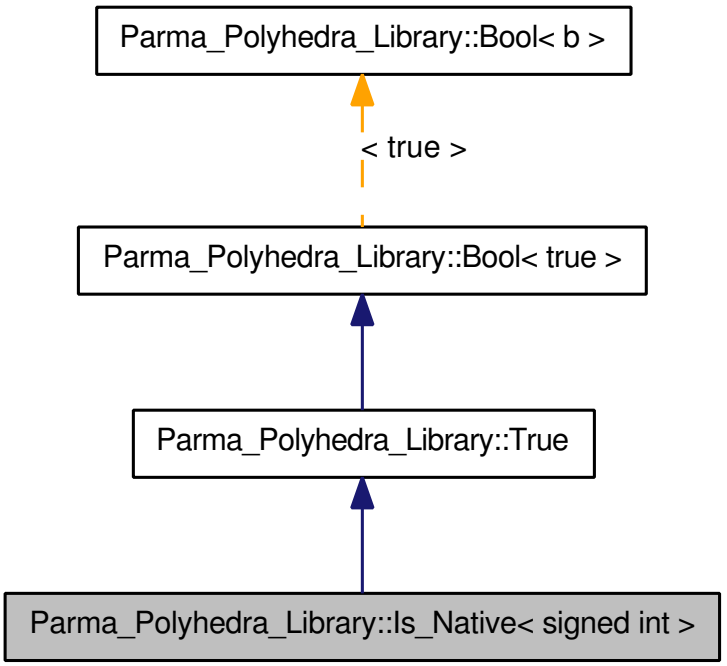
11.100 Parma_Polyhedra_Library::Is_Native< signed int > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< signed int >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< signed int >:



11.100.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< signed int >

Definition at line 317 of file meta_programming.hh.

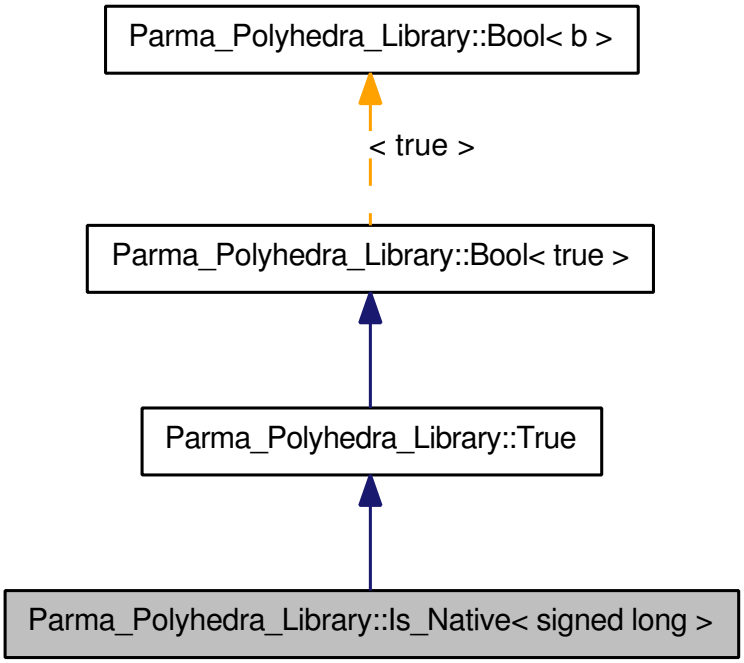
The documentation for this struct was generated from the following file:

- meta_programming.hh

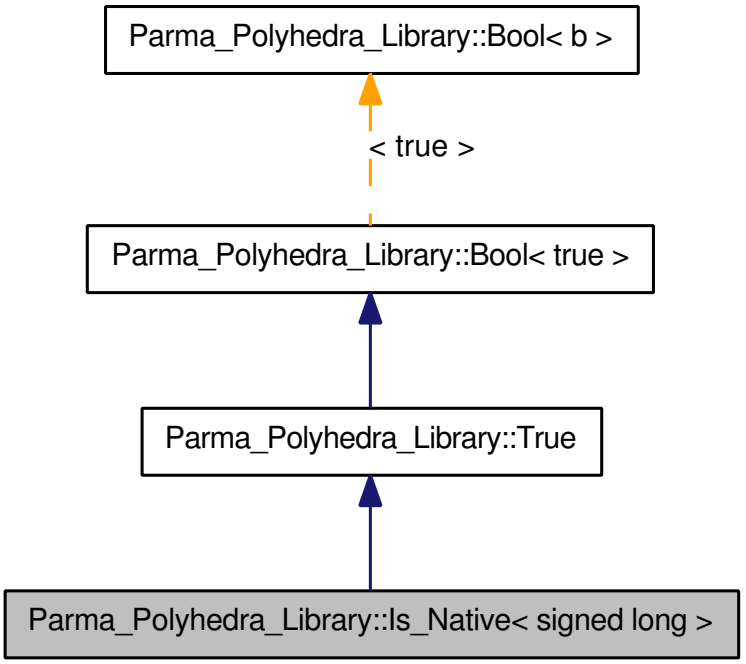
11.101 Parma_Polyhedra_Library::Is_Native< signed long > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< signed long >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< signed long >:



11.101.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< signed long >

Definition at line 318 of file meta_programming.hh.

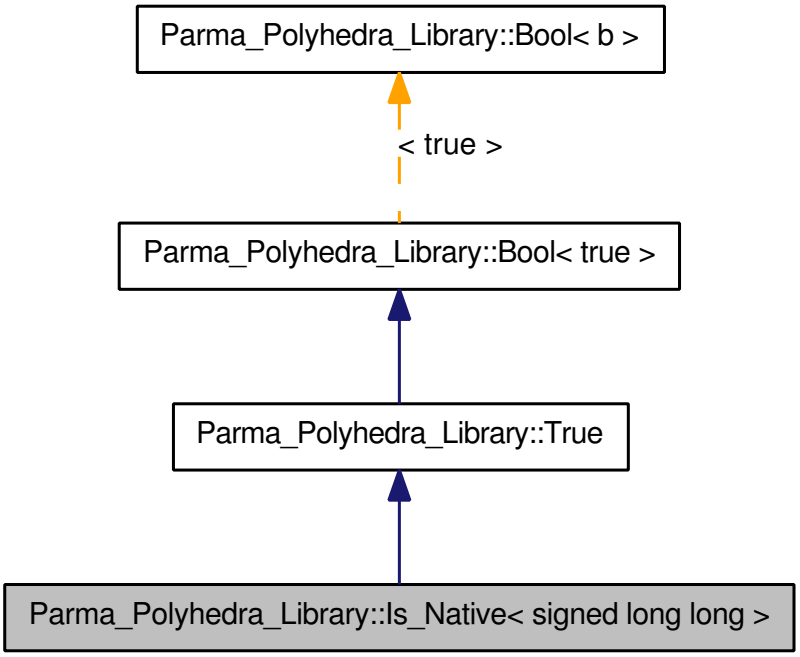
The documentation for this struct was generated from the following file:

- meta_programming.hh

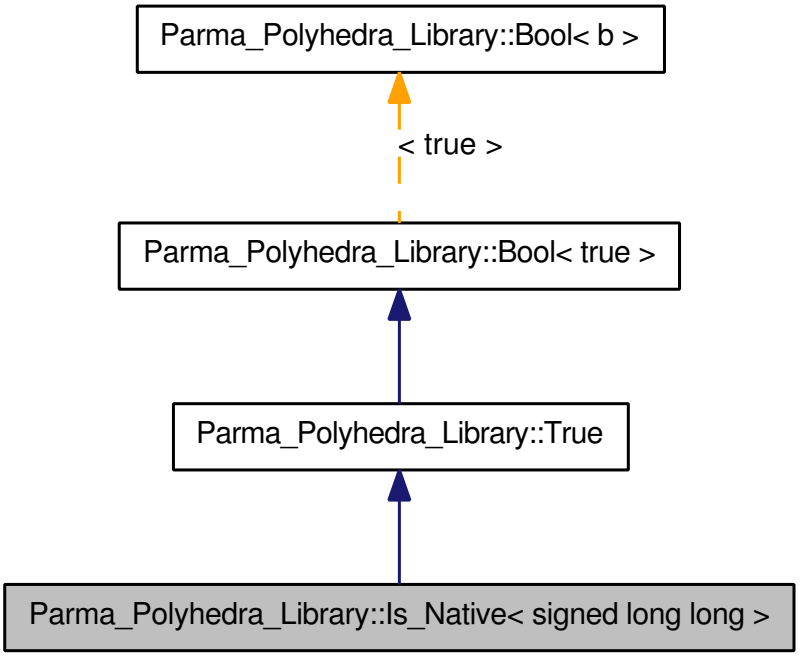
11.102 Parma_Polyhedra_Library::Is_Native< signed long long > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< signed long long >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< signed long long >:



11.102.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< signed long long >

Definition at line 319 of file meta_programming.hh.

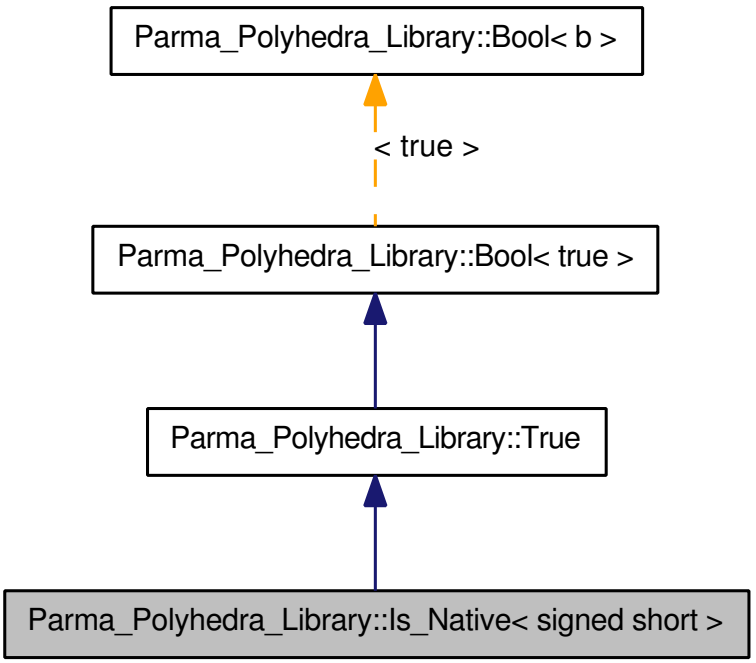
The documentation for this struct was generated from the following file:

- meta_programming.hh

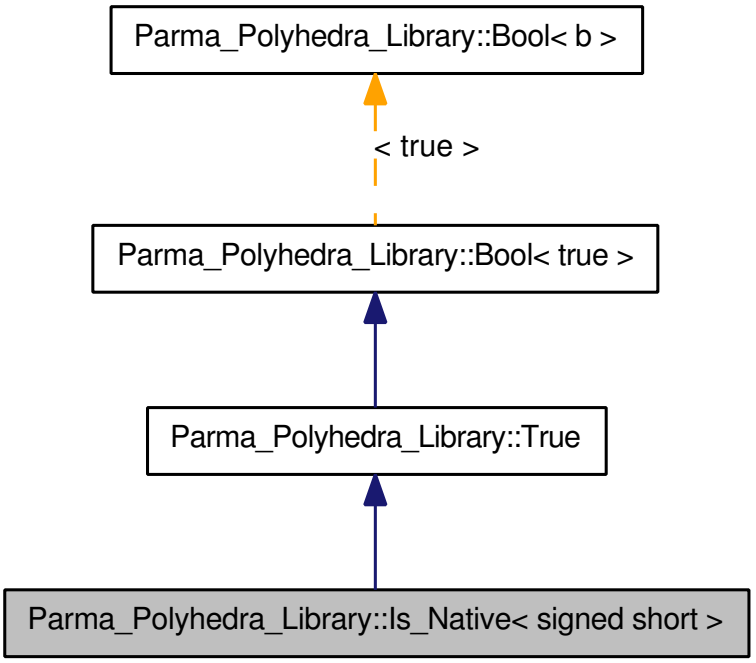
11.103 Parma_Polyhedra_Library::Is_Native< signed short > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< signed short >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< signed short >:



11.103.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< signed short >

Definition at line 316 of file meta_programming.hh.

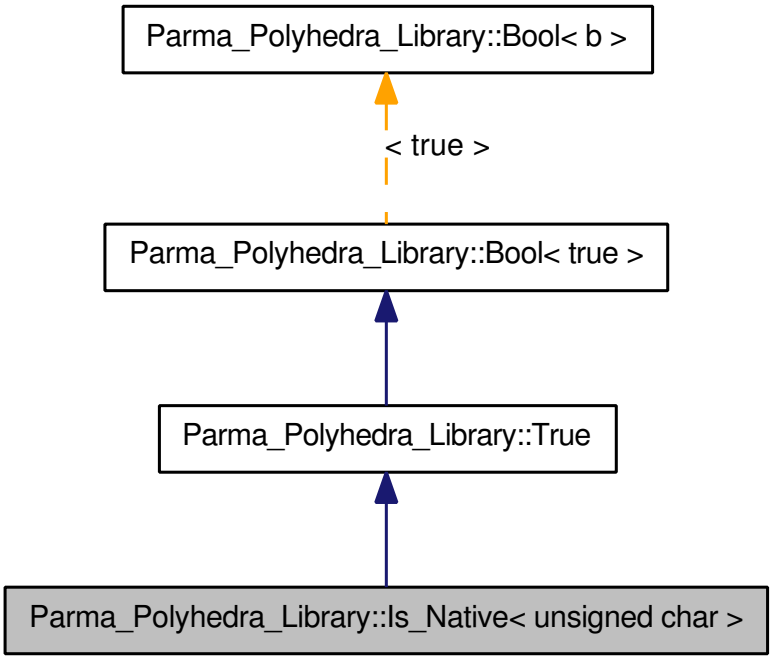
The documentation for this struct was generated from the following file:

- meta_programming.hh

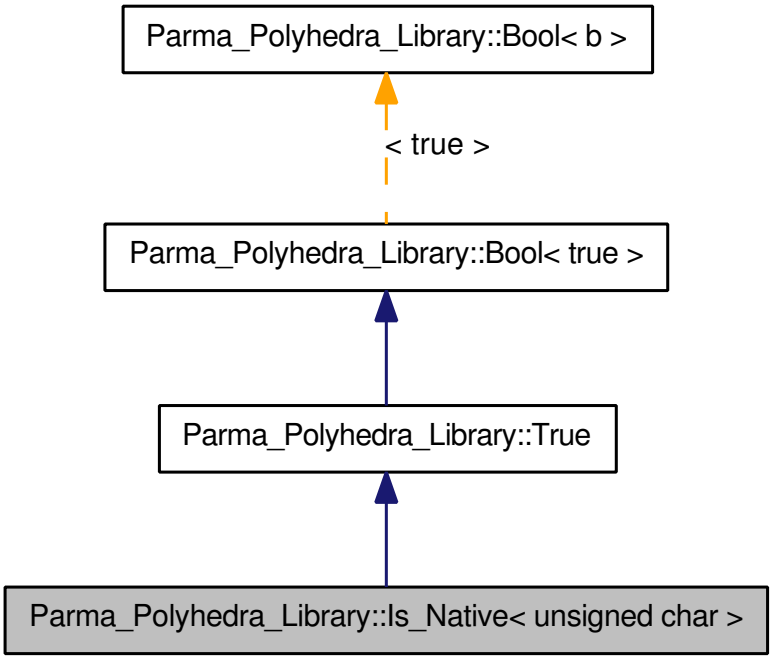
11.104 Parma_Polyhedra_Library::Is_Native< unsigned char > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< unsigned char >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< unsigned char >:



11.104.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< unsigned char >

Definition at line 320 of file meta_programming.hh.

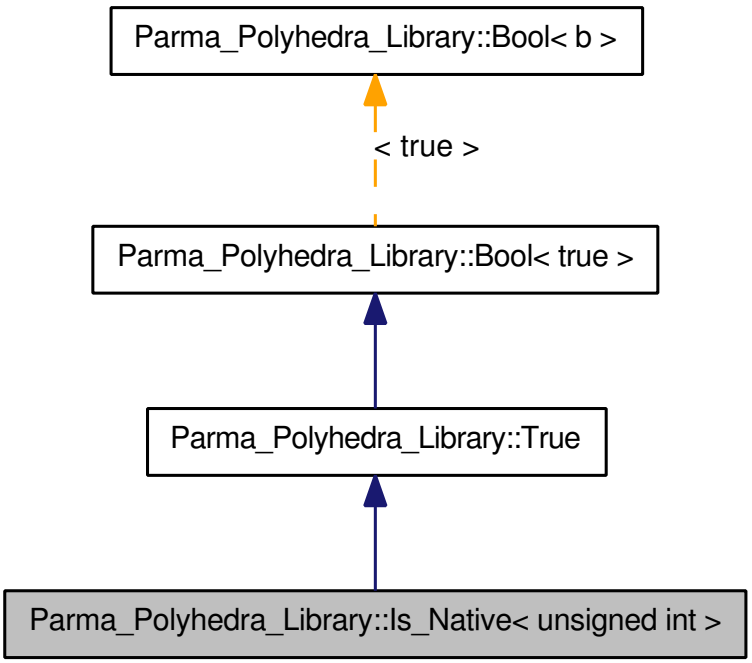
The documentation for this struct was generated from the following file:

- meta_programming.hh

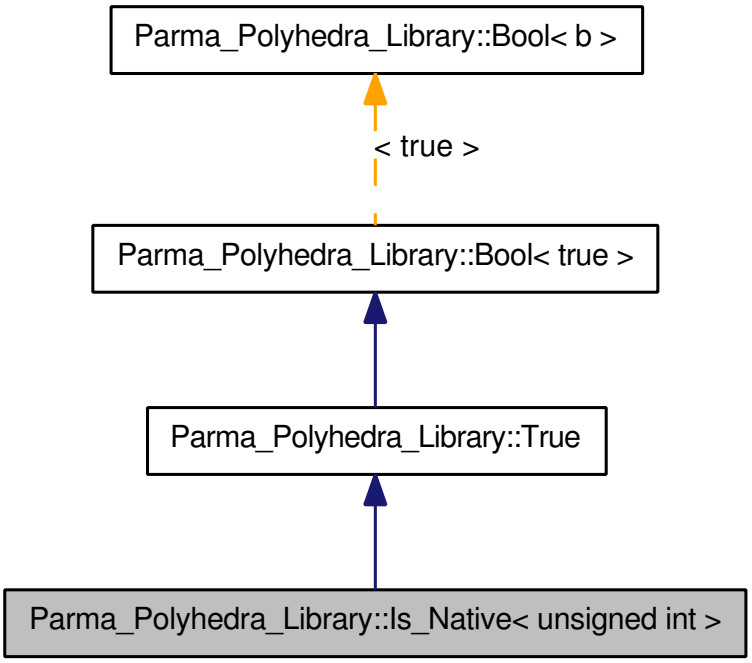
11.105 Parma_Polyhedra_Library::Is_Native< unsigned int > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< unsigned int >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< unsigned int >:



11.105.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< unsigned int >

Definition at line 322 of file meta_programming.hh.

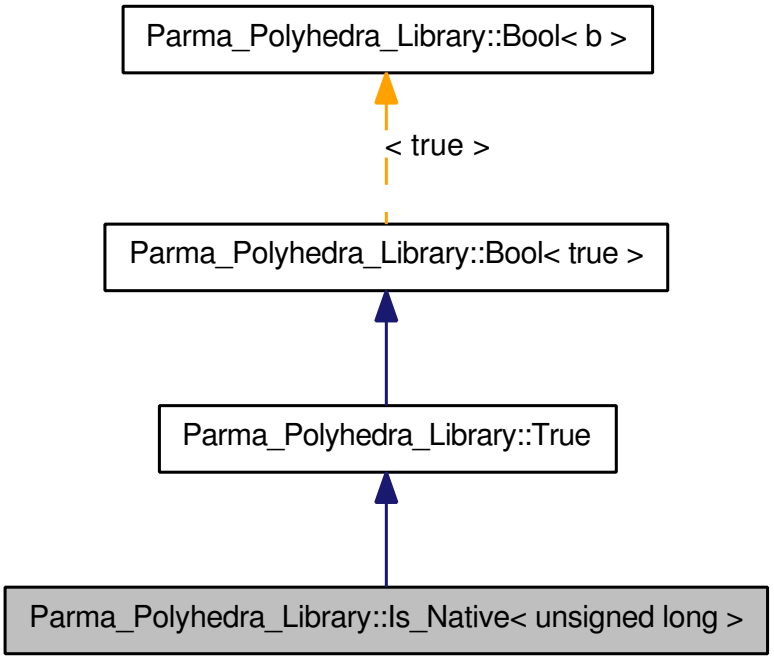
The documentation for this struct was generated from the following file:

- meta_programming.hh

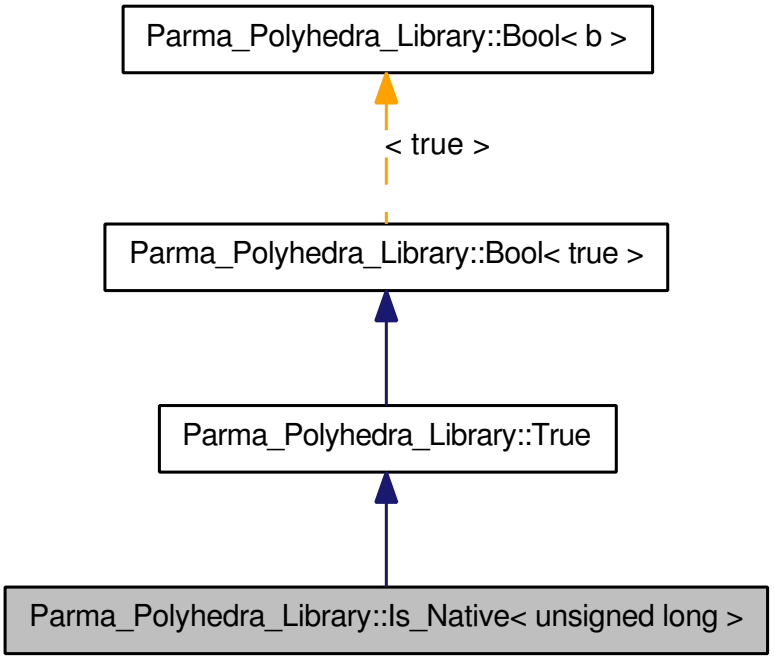
11.106 Parma_Polyhedra_Library::Is_Native< unsigned long > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< unsigned long >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< unsigned long >:



11.106.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< unsigned long >

Definition at line 323 of file meta_programming.hh.

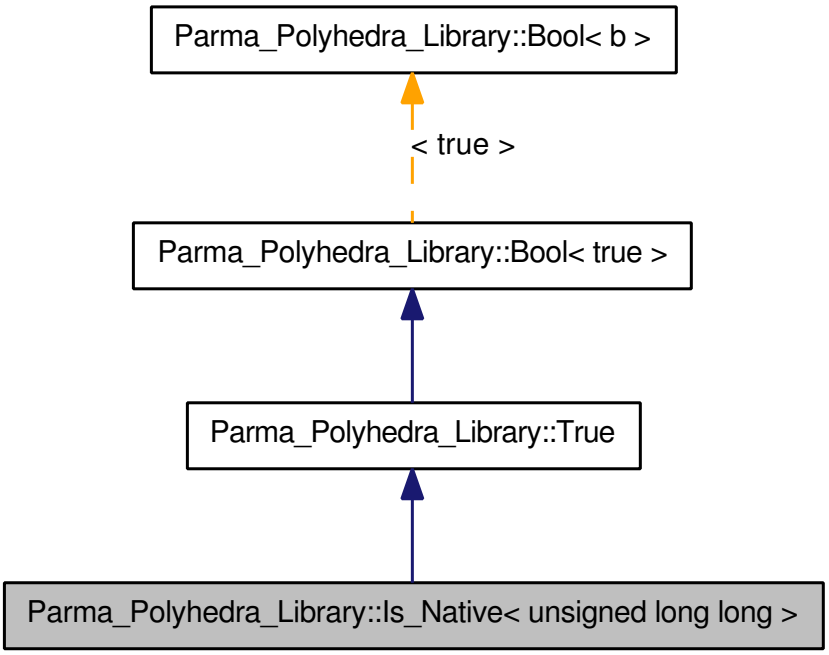
The documentation for this struct was generated from the following file:

- meta_programming.hh

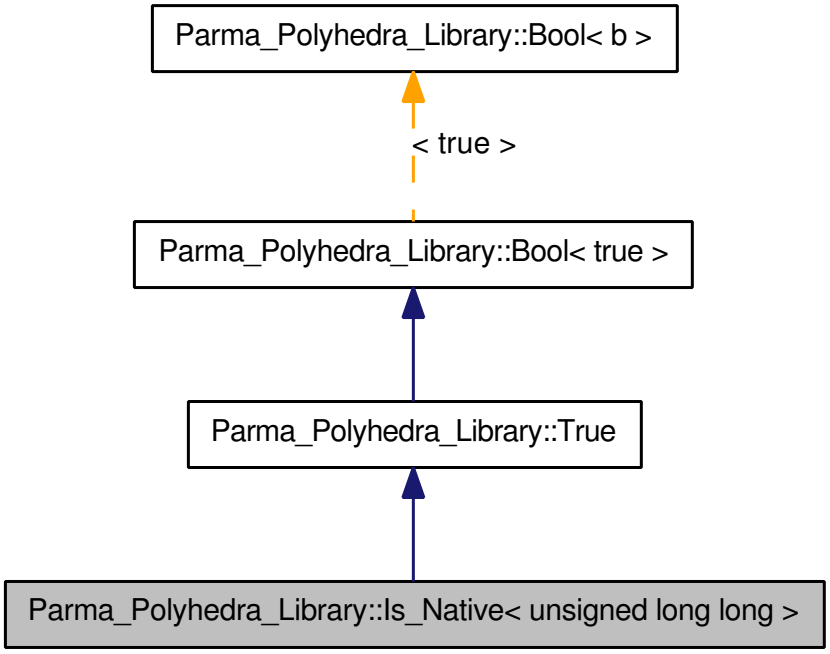
11.107 Parma_Polyhedra_Library::Is_Native< unsigned long long > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< unsigned long long >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< unsigned long long >:



11.107.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< unsigned long long >

Definition at line 324 of file meta_programming.hh.

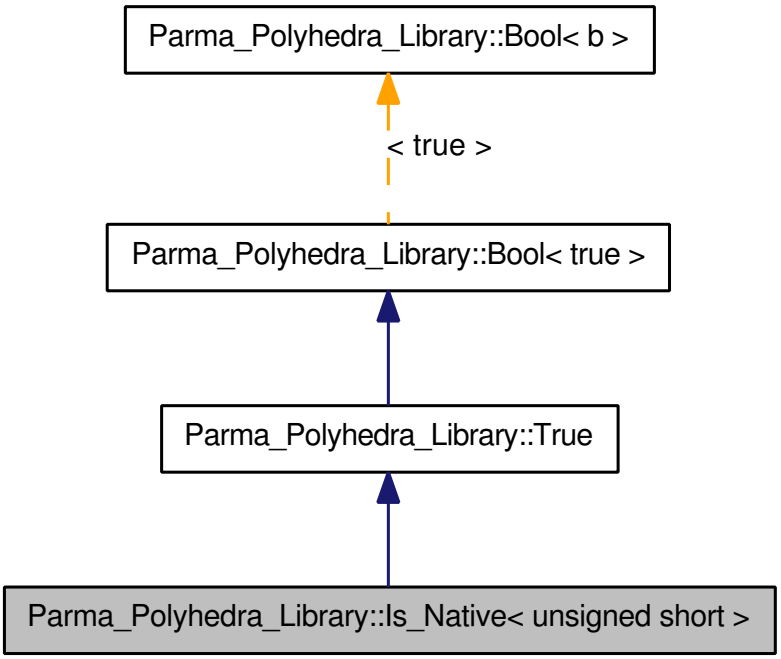
The documentation for this struct was generated from the following file:

- meta_programming.hh

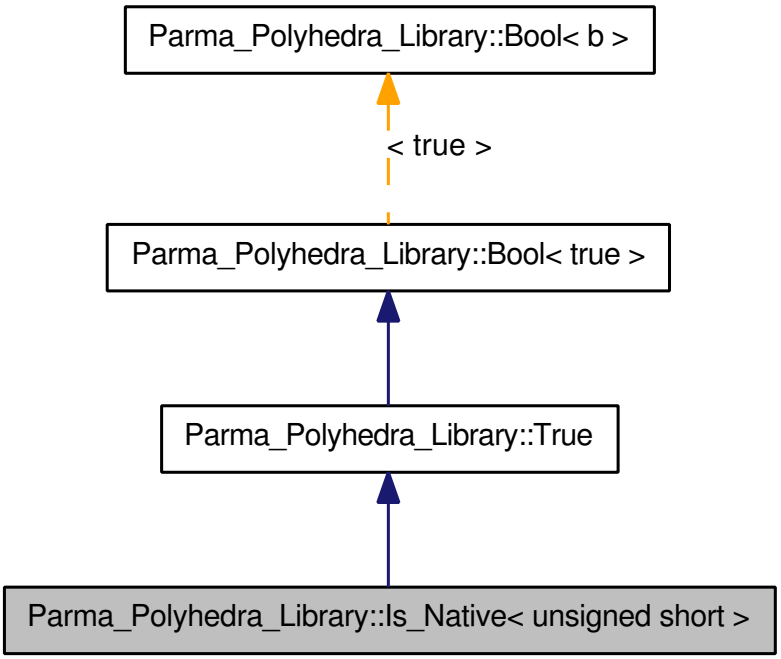
11.108 Parma_Polyhedra_Library::Is_Native< unsigned short > Struct Template Reference

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native< unsigned short >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native< unsigned short >:



11.108.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Native< unsigned short >

Definition at line 321 of file meta_programming.hh.

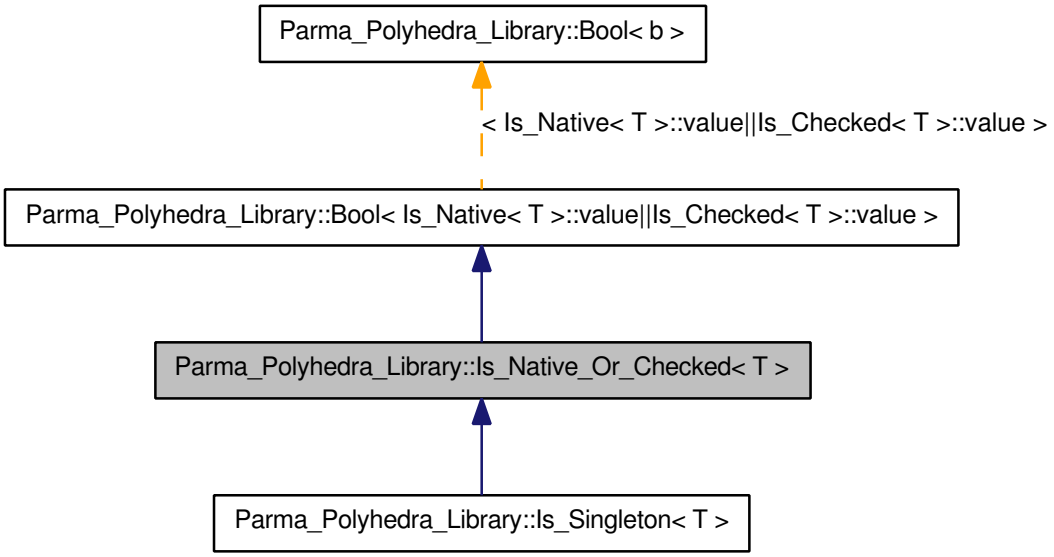
The documentation for this struct was generated from the following file:

- meta_programming.hh

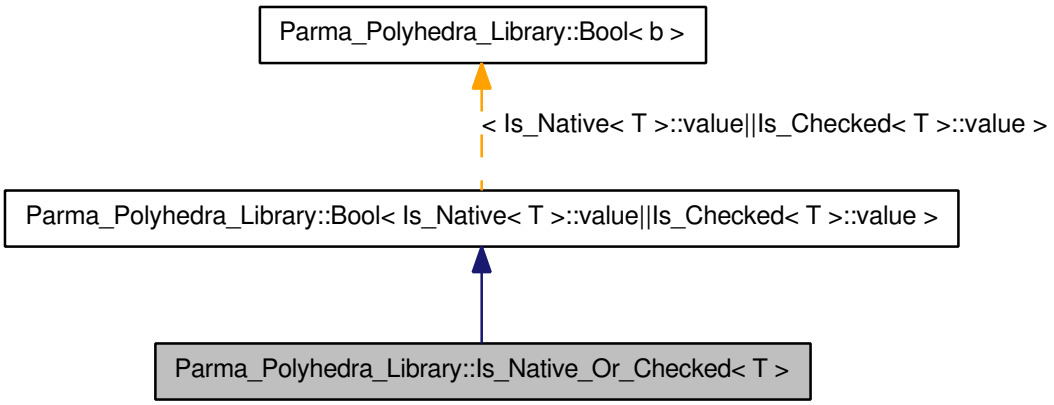
11.109 Parma_Polyhedra_Library::Is_Native_Or_Checked< T > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Native_Or_Checked< T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Native_Or_Checked< T >:



11.109.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Native_Or_Checked< T >

Definition at line 147 of file Checked_Number.defs.hh.

The documentation for this struct was generated from the following file:

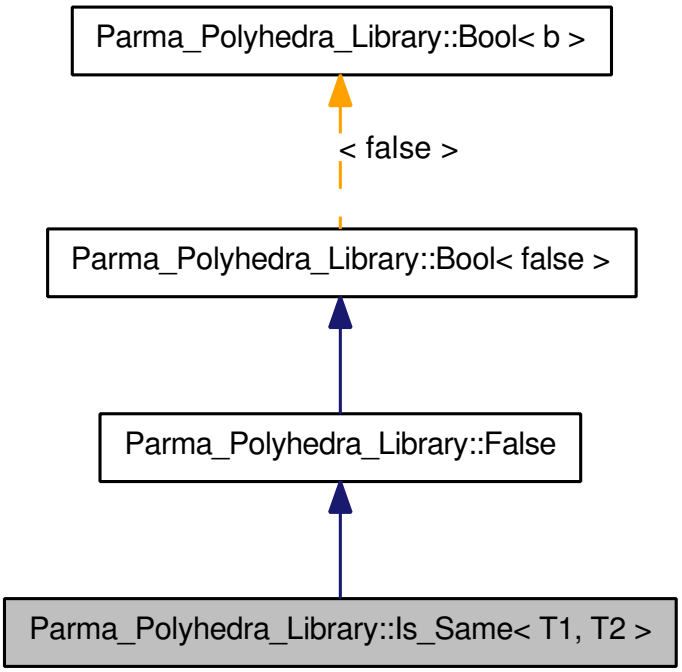
- Checked_Number.defs.hh

11.110 Parma_Polyhedra_Library::Is_Same< T1, T2 > Struct Template Reference

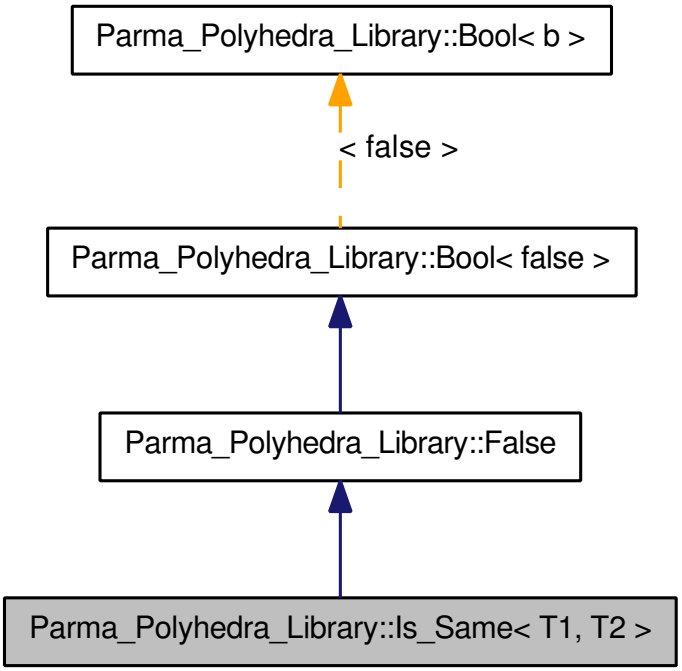
A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Same< T1, T2 >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Same< T1, T2 >:



11.110.1 Detailed Description

template<typename T1, typename T2> struct Parma_Polyhedra_Library::Is_Same< T1, T2 >

A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`. This is the non-specialized case, in which `T1` and `T2` can be different.

Definition at line 180 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

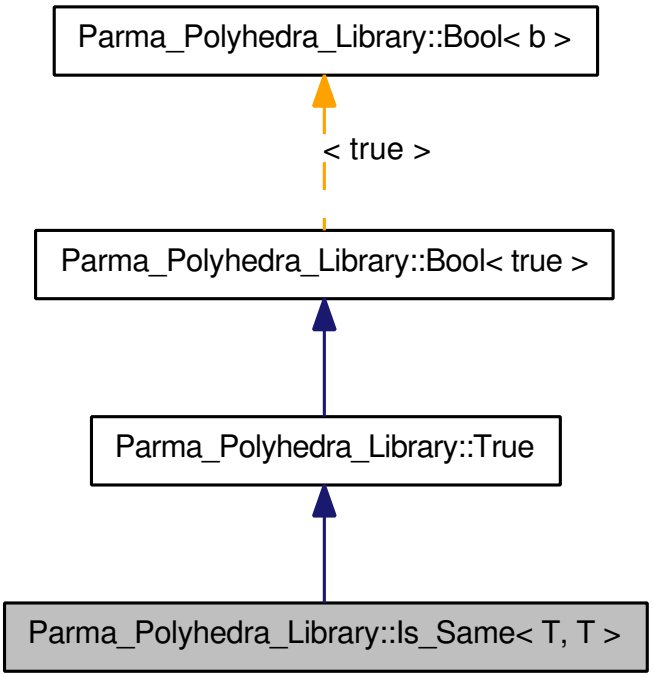
- meta_programming.hh

11.111 Parma_Polyhedra_Library::Is_Same< T, T > Struct Template Reference

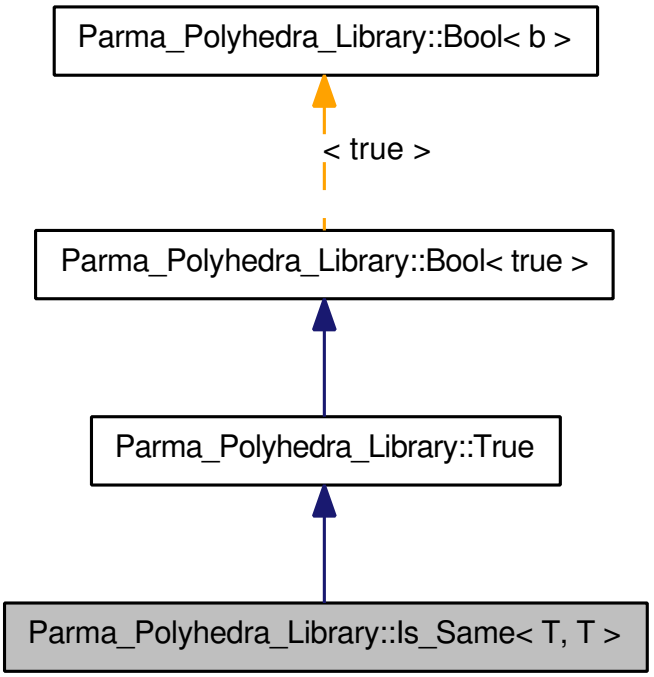
A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Same< T, T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Same< T, T >:



11.111.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Same< T, T >

A class holding a constant called `value` that evaluates to `true` if and only if `T1` is the same type as `T2`. This is the specialization in which `T1` and `T2` are equal.

Definition at line 193 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

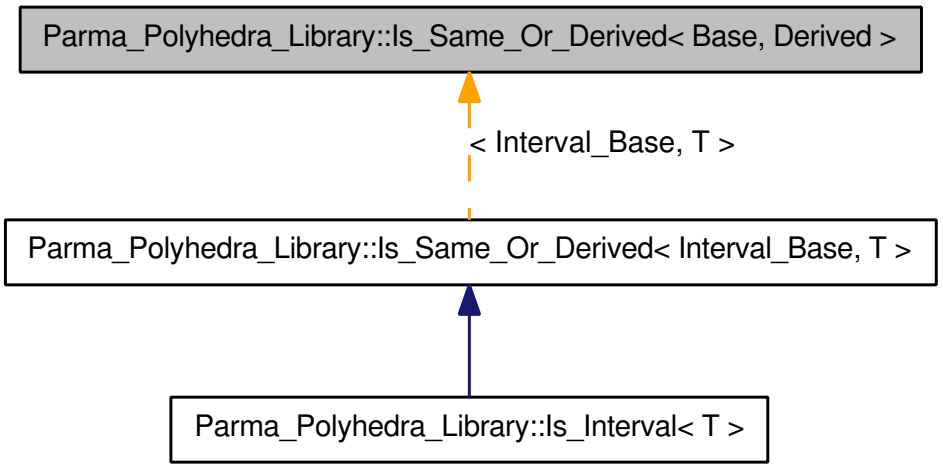
- meta_programming.hh

11.112 Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived > Struct Template Reference

A class holding a constant called `value` that evaluates to `true` if and only if `Base` is the same type as `Derived` or `Derived` is a class derived from `Base`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >:



Classes

- struct Any

  A class that is constructible from just anything.

Public Types

- enum anonymous_enum { value = (sizeof(func(derived_object())) == sizeof(char)) }

Public Member Functions

- PPL_COMPILE_TIME_CHECK (sizeof(char)!=sizeof(double),"architecture with sizeof(char) == sizeof(double)"" (!?)")

Static Public Member Functions

- static char func (const Base &)

  Overloading with `Base`.

- static double func (Any)

  Overloading with `Any`.

- static const Derived & derived_object ()

  A function obtaining a const reference to a `Derived` *object.*

11.112.1 Detailed Description

template<typename Base, typename Derived> struct Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >

A class holding a constant called `value` that evaluates to `true` if and only if `Base` is the same type as `Derived` or `Derived` is a class derived from `Base`.

Note

Care must be taken to use this predicate with template classes. Suppose we have

```
template <typename T> struct B;
template <typename T> struct D : public B<T>;
```

Of course, we cannot test if, for some type variable `U`, we have `Is_Same_Or_Derived<B<U>, Type>::anonymous_enum::value == true`. But we can do as follows:

```
struct B_Base {
};

template <typename T> struct B : public B_Base;
```

This enables us to enquire `Is_Same_Or_Derived<B_Base, Type>::anonymous_enum::value`.

Definition at line 224 of file meta_programming.hh.

11.112.2 Member Enumeration Documentation

11.112.2.1 template<typename Base, typename Derived> enum Parma_Polyhedra_Library::Is_Same_Or_Derived::anonymous_enum

Enumerator:

value Assuming `sizeof(char) != sizeof(double)`, the C++ overload resolution mechanism guarantees that `value` evaluates to `true` if and only if `Base` is the same type as `Derived` or `Derived` is a class derived from `Base`.

Definition at line 248 of file meta_programming.hh.

11.112.3 Member Function Documentation

11.112.3.1 template<typename Base, typename Derived> static const Derived& Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::derived_object () [static]

A function obtaining a const reference to a `Derived` object.

11.112.3.2 template<typename Base, typename Derived> static double Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::func (Any) [static]

Overloading with `Any`.

11.112.3.3 template<typename Base, typename Derived> static char Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::func (const Base &) [static]

Overloading with `Base`.

11.112.3.4 template<typename Base, typename Derived> Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::PPL_COMPILE_TIME_CHECK (sizeof(char)! = sizeof(double))

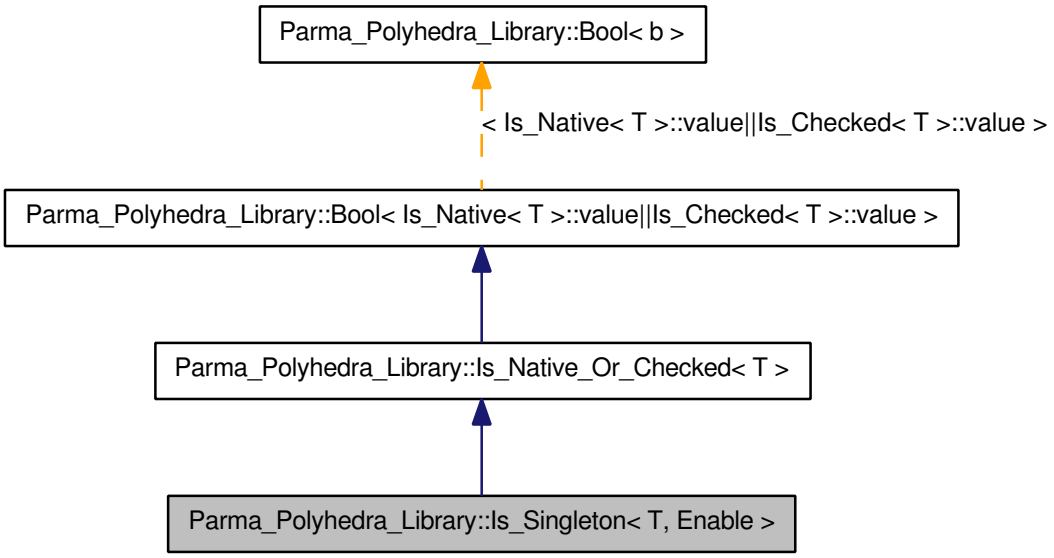
The documentation for this struct was generated from the following file:

- meta_programming.hh

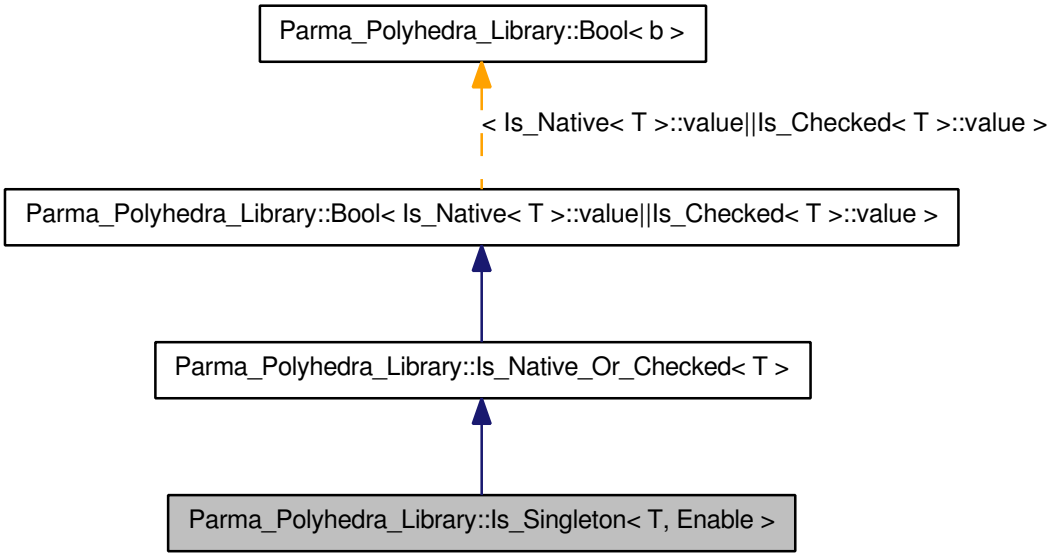
11.113 Parma_Polyhedra_Library::Is_Singleton< T, Enable > Struct Template Reference

`#include <Interval.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Singleton< T, Enable >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Singleton< T, Enable >:



11.113.1 Detailed Description

template<typename T, typename Enable = void> struct Parma_Polyhedra_Library::Is_Singleton< T, Enable >

Definition at line 54 of file Interval.defs.hh.

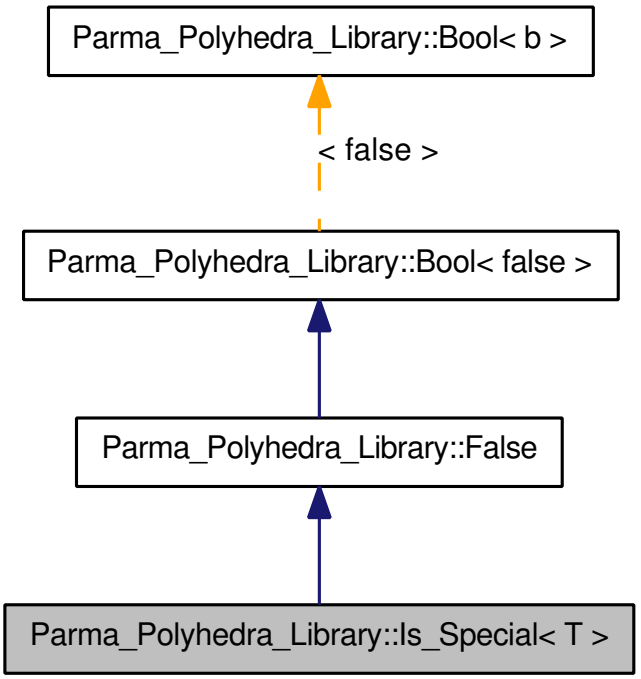
The documentation for this struct was generated from the following file:

- Interval.defs.hh

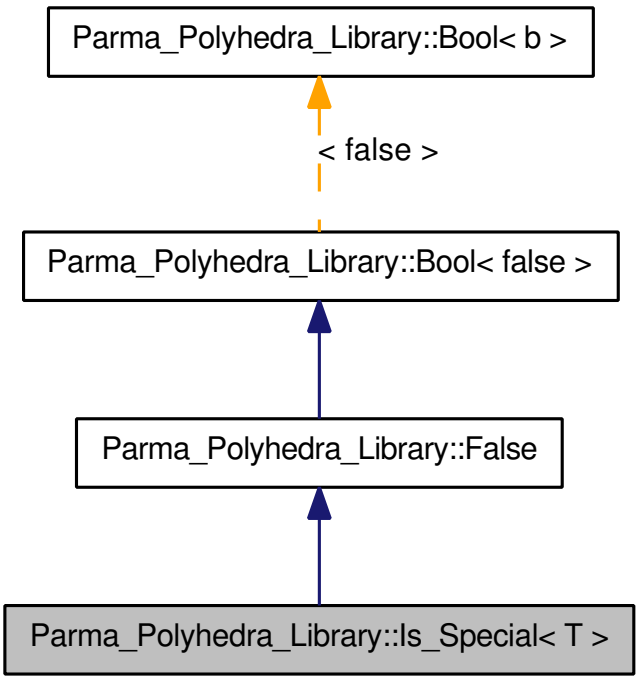
11.114 Parma_Polyhedra_Library::Is_Special< T > Struct Template Reference

`#include <checked.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Special< T >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Special< T >:



11.114.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Is_Special< T >

Definition at line 551 of file checked.defs.hh.

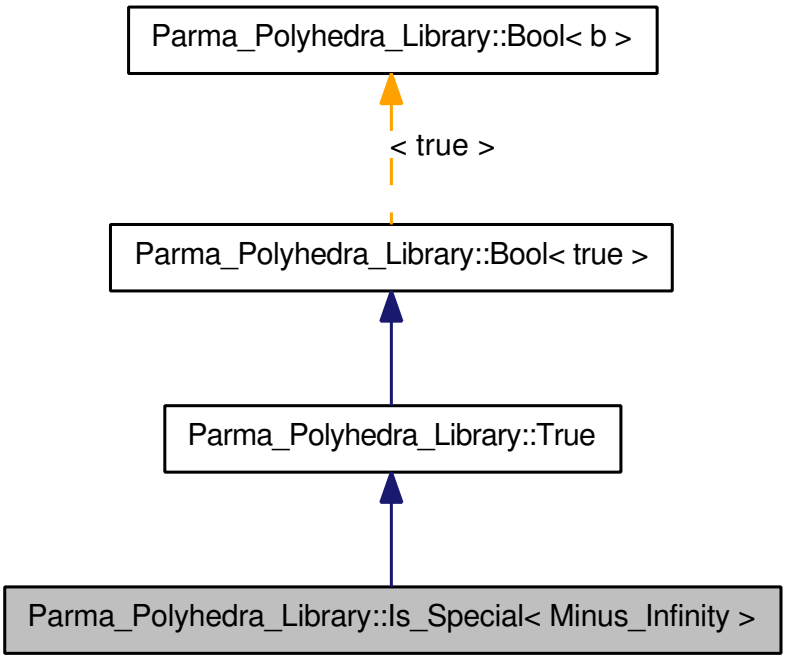
The documentation for this struct was generated from the following file:

- checked.defs.hh

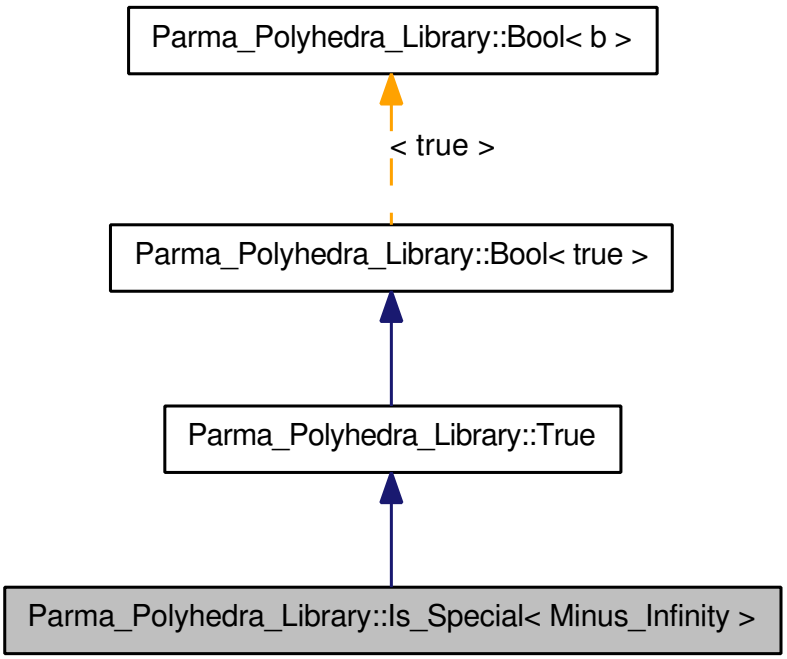
11.115 Parma_Polyhedra_Library::Is_Special< Minus_Infinity > Struct Template Reference

`#include <checked.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Special< Minus_Infinity >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Special< Minus_Infinity >:



11.115.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Special< Minus_Infinity >

Definition at line 554 of file checked.defs.hh.

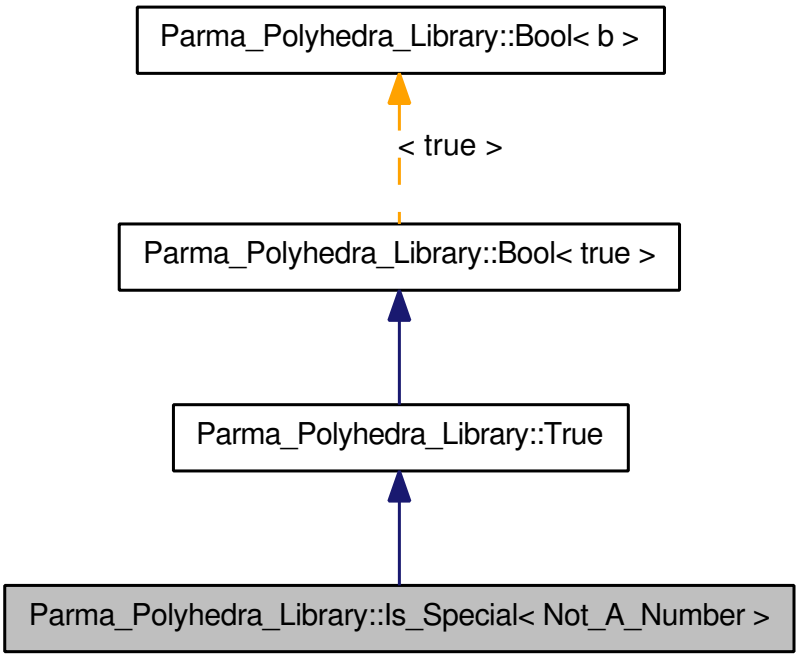
The documentation for this struct was generated from the following file:

- checked.defs.hh

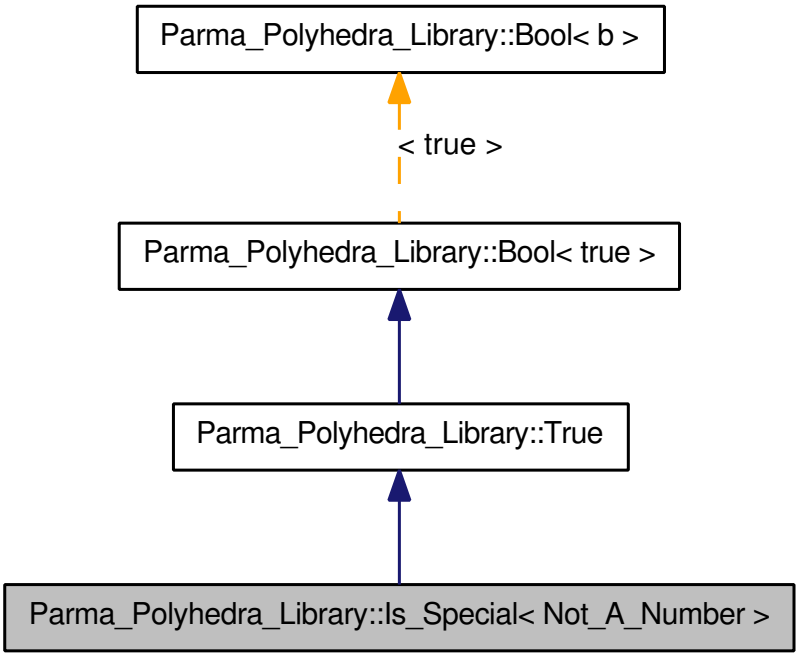
11.116 Parma_Polyhedra_Library::Is_Special< Not_A_Number > Struct Template Reference

`#include <checked.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Special< Not_A_Number >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Special< Not_A_Number >:



11.116.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Special< Not_A_Number >

Definition at line 560 of file checked.defs.hh.

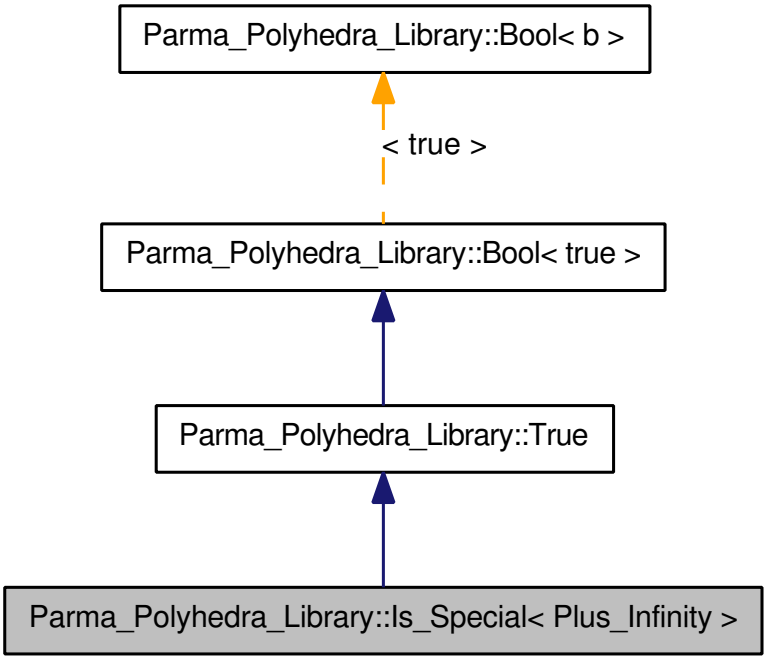
The documentation for this struct was generated from the following file:

- checked.defs.hh

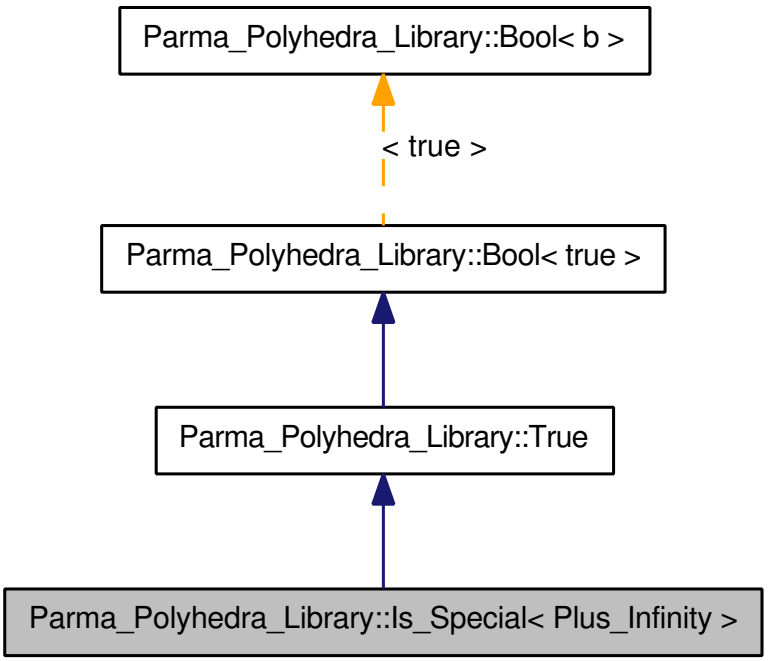
11.117 Parma_Polyhedra_Library::Is_Special< Plus_Infinity > Struct Template Reference

`#include <checked.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Is_Special< Plus_Infinity >:



Collaboration diagram for Parma_Polyhedra_Library::Is_Special< Plus_Infinity >:



11.117.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Is_Special< Plus_Infinity >

Definition at line 557 of file checked.defs.hh.

The documentation for this struct was generated from the following file:

- checked.defs.hh

11.118 Parma_Polyhedra_Library::iterator_to_const< Container > Class Template Reference

An iterator on a sequence of read-only objects.

`#include <iterator_to_const.defs.hh>`

Public Types

- typedef Traits::iterator_category iterator_category
- typedef Traits::value_type value_type
- typedef Traits::difference_type difference_type
- typedef Traits::pointer pointer
- typedef Traits::reference reference

Public Member Functions

- iterator_to_const ()
  Default constructor.
- iterator_to_const (const iterator_to_const &y)
  Copy constructor.
- reference operator* () const
  Dereference operator.
- pointer operator-> () const
  Indirect access operator.
- iterator_to_const & operator++ ()
  Prefix increment operator.
- iterator_to_const operator++ (int)
  Postfix increment operator.
- iterator_to_const & operator-- ()
  Prefix decrement operator.
- iterator_to_const operator-- (int)
  Postfix decrement operator.

- bool operator== (const iterator_to_const &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are identical.*

- bool operator!= (const iterator_to_const &y) const

  Returns `true` *if and only if* `*this` *and* `y` *are different.*

Private Types

- typedef Container::iterator Base

  The type of the underlying mutable iterator.

- typedef std::iterator_traits< typename Container::const_iterator > Traits

  A shortcut for naming the const_iterator traits.

Private Member Functions

- iterator_to_const (const Base &b)

  Constructs from the lower-level iterator.

Private Attributes

- Base base

  A (mutable) iterator on the sequence of elements.

Friends

- class const_iterator_to_const< Container >
- class Powerset

11.118.1 Detailed Description

template<typename Container> class Parma_Polyhedra_Library::iterator_to_const< Container >

An iterator on a sequence of read-only objects. This template class implements a bidirectional *read-only* iterator on the sequence of objects `Container`. By using this iterator class it is not possible to modify the objects contained in `Container`; rather, object modification has to be implemented by object replacement, i.e., by using the methods provided by `Container` to remove/insert objects. Such a policy (a modifiable container of read-only objects) allows for a reliable enforcement of invariants (such as sortedness of the objects in the sequence).

Note

For any developers' need, suitable friend declarations allow for accessing the low-level iterators on the sequence of objects.

Definition at line 51 of file iterator_to_const.defs.hh.

11.118.2 Member Typedef Documentation

11.118.2.1 template<typename Container> typedef Container::iterator Parma_Polyhedra_Library::iterator_to_const< Container >::Base [private]

The type of the underlying mutable iterator.

Definition at line 54 of file iterator_to_const.defs.hh.

11.118.2.2 template<typename Container> typedef Traits::difference_type Parma_Polyhedra_Library::iterator_to_const< Container >::difference_type

Definition at line 75 of file iterator_to_const.defs.hh.

11.118.2.3 template<typename Container> typedef Traits::iterator_category Parma_Polyhedra_Library::iterator_to_const< Container >::iterator_category

Definition at line 73 of file iterator_to_const.defs.hh.

11.118.2.4 template<typename Container> typedef Traits::pointer Parma_Polyhedra_Library::iterator_to_const< Container >::pointer

Definition at line 76 of file iterator_to_const.defs.hh.

11.118.2.5 template<typename Container> typedef Traits::reference Parma_Polyhedra_Library::iterator_to_const< Container >::reference

Definition at line 77 of file iterator_to_const.defs.hh.

11.118.2.6 template<typename Container> typedef std::iterator_traits<typename Container::const_iterator> Parma_Polyhedra_Library::iterator_to_const< Container >::Traits [private]

A shortcut for naming the const_iterator traits.

Definition at line 58 of file iterator_to_const.defs.hh.

11.118.2.7 template<typename Container> typedef Traits::value_type Parma_Polyhedra_Library::iterator_to_const< Container >::value_type

Definition at line 74 of file iterator_to_const.defs.hh.

11.118.3 Constructor & Destructor Documentation

11.118.3.1 template<typename Container > Parma_Polyhedra_Library::iterator_to_const< Container >::iterator_to_const (const Base & *b*) [inline, private]

Constructs from the lower-level iterator.

Definition at line 44 of file iterator_to_const.inlines.hh.

11.118.3.2 template<typename Container > Parma_Polyhedra_Library::iterator_to_const< Container >::iterator_to_const () [inline]

Default constructor.

Definition at line 32 of file iterator_to_const.inlines.hh.

11.118.3.3 template<typename Container > Parma_Polyhedra_Library::iterator_to_const< Container >::iterator_to_const (const iterator_to_const< Container > & *y*) [inline]

Copy constructor.

Definition at line 38 of file iterator_to_const.inlines.hh.

11.118.4 Member Function Documentation

11.118.4.1 template<typename Container > bool Parma_Polyhedra_Library::iterator_to_const< Container >::operator!= (const iterator_to_const< Container > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are different.

Definition at line 98 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::operator==().

11.118.4.2 template<typename Container > iterator_to_const< Container >::reference Parma_Polyhedra_Library::iterator_to_const< Container >::operator* () const [inline]

Dereference operator.

Definition at line 50 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base.

11.118.4.3 template<typename Container > iterator_to_const< Container > Parma_Polyhedra_Library::iterator_to_const< Container >::operator++ (int) [inline]

Postfix increment operator.

Definition at line 69 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::operator++().

11.118.4.4 template<typename Container > iterator_to_const< Container > & Parma_Polyhedra_Library::iterator_to_const< Container >::operator++ () [inline]

Prefix increment operator.

Definition at line 62 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base.

Referenced by Parma_Polyhedra_Library::iterator_to_const< Container >::operator++().

11.118.4.5 template<typename Container > iterator_to_const< Container > Parma_Polyhedra_Library::iterator_to_const< Container >::operator-- (int) [inline]

Postfix decrement operator.

Definition at line 84 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::operator--().

11.118.4.6 template<typename Container > iterator_to_const< Container > & Parma_Polyhedra_Library::iterator_to_const< Container >::operator-- () [inline]

Prefix decrement operator.

Definition at line 77 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base.

Referenced by Parma_Polyhedra_Library::iterator_to_const< Container >::operator--().

11.118.4.7 template<typename Container > iterator_to_const< Container >::pointer Parma_Polyhedra_Library::iterator_to_const< Container >::operator-> () const [inline]

Indirect access operator.

Definition at line 56 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base.

11.118.4.8 template<typename Container > bool Parma_Polyhedra_Library::iterator_to_const< Container >::operator== (const iterator_to_const< Container > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are identical.

Definition at line 92 of file iterator_to_const.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base.

Referenced by Parma_Polyhedra_Library::iterator_to_const< Container >::operator!=().

11.118.5 Friends And Related Function Documentation

11.118.5.1 template<typename Container> friend class const_iterator_to_const< Container > [friend]

Definition at line 66 of file iterator_to_const.defs.hh.

11.118.5.2 template<typename Container> friend class Powerset [friend]

Definition at line 67 of file iterator_to_const.defs.hh.

11.118.6 Member Data Documentation

11.118.6.1 template<typename Container> Base Parma_Polyhedra_Library::iterator_to_const< Container >::base [private]

A (mutable) iterator on the sequence of elements.

Definition at line 61 of file iterator_to_const.defs.hh.

Referenced by Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::drop_disjuncts(), Parma_Polyhedra_Library::iterator_to_const< Container >::operator*(), Parma_Polyhedra_Library::iterator_to_const< Container >::operator++(), Parma_Polyhedra_Library::iterator_to_const<

Container >::operator--(), Parma_Polyhedra_Library::iterator_to_const< Container >::operator->(), and Parma_Polyhedra_Library::iterator_to_const< Container >::operator==().

The documentation for this class was generated from the following files:

- iterator_to_const.defs.hh
- iterator_to_const.inlines.hh

11.119 Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp > Struct Template Reference

`#include <distances.defs.hh>`

Static Public Member Functions

- static void combine (Temp &running, const Temp &current, Rounding_Dir)
- static void finalize (Temp &, Rounding_Dir)

11.119.1 Detailed Description

template<typename Temp> struct Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp >

Definition at line 46 of file distances.defs.hh.

11.119.2 Member Function Documentation

11.119.2.1 template<typename Temp > void Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp >::combine (Temp & *running*, const Temp & *current*, Rounding_Dir) [inline, static]

Definition at line 99 of file distances.inlines.hh.

11.119.2.2 template<typename Temp > void Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp >::finalize (Temp &, Rounding_Dir) [inline, static]

Definition at line 108 of file distances.inlines.hh.

The documentation for this struct was generated from the following files:

- distances.defs.hh
- distances.inlines.hh

11.120 Parma_Polyhedra_Library::Checked::Larger< char > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast16_t type_for_neg
- typedef int_fast16_t type_for_add
- typedef int_fast16_t type_for_sub
- typedef int_fast16_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, true)
- const_bool_nodef (use_for_add, true)
- const_bool_nodef (use_for_sub, true)
- const_bool_nodef (use_for_mul, true)

11.120.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< char >

Definition at line 789 of file checked_int.inlines.hh.

11.120.2 Member Typedef Documentation

11.120.2.1 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< char >::type_for_add

Definition at line 795 of file checked_int.inlines.hh.

11.120.2.2 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< char >::type_for_mul

Definition at line 797 of file checked_int.inlines.hh.

11.120.2.3 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< char >::type_for_neg

Definition at line 794 of file checked_int.inlines.hh.

11.120.2.4 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< char >::type_for_sub

Definition at line 796 of file checked_int.inlines.hh.

11.120.3 Member Function Documentation

11.120.3.1 Parma_Polyhedra_Library::Checked::Larger< char >::const_bool_nodef (use_for_mul, true)

11.120.3.2 Parma_Polyhedra_Library::Checked::Larger< char >::const_bool_nodef (use_for_sub, true)

11.120.3.3 Parma_Polyhedra_Library::Checked::Larger< char >::const_bool_nodef (use_for_add, true)

11.120.3.4 Parma_Polyhedra_Library::Checked::Larger< char >::const_bool_nodef (use_for_neg, true)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.121 Parma_Polyhedra_Library::Checked::Larger< signed char > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast16_t type_for_neg
- typedef int_fast16_t type_for_add
- typedef int_fast16_t type_for_sub
- typedef int_fast16_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, true)
- const_bool_nodef (use_for_add, true)
- const_bool_nodef (use_for_sub, true)
- const_bool_nodef (use_for_mul, true)

11.121.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< signed char >

Definition at line 801 of file checked_int.inlines.hh.

11.121.2 Member Typedef Documentation

11.121.2.1 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< signed char >::type_for_add

Definition at line 807 of file checked_int.inlines.hh.

11.121.2.2 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< signed char >::type_for_mul

Definition at line 809 of file checked_int.inlines.hh.

11.121.2.3 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< signed char >::type_for_neg

Definition at line 806 of file checked_int.inlines.hh.

11.121.2.4 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< signed char >::type_for_sub

Definition at line 808 of file checked_int.inlines.hh.

11.121.3 Member Function Documentation

11.121.3.1 Parma_Polyhedra_Library::Checked::Larger< signed char >::const_bool_nodef (use_for_mul, true)

11.121.3.2 Parma_Polyhedra_Library::Checked::Larger< signed char >::const_bool_nodef (use_for_sub, true)

11.121.3.3 Parma_Polyhedra_Library::Checked::Larger< signed char >::const_bool_nodef (use_for_add, true)

11.121.3.4 Parma_Polyhedra_Library::Checked::Larger< signed char >::const_bool_nodef (use_for_neg, true)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.122 Parma_Polyhedra_Library::Checked::Larger< signed int > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef int_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef int_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg,(LONG_BITS==64))
- const_bool_nodef (use_for_add,(LONG_BITS==64))
- const_bool_nodef (use_for_sub,(LONG_BITS==64))
- const_bool_nodef (use_for_mul, true)

11.122.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< signed int >

Definition at line 849 of file checked_int.inlines.hh.

11.122.2 Member Typedef Documentation

11.122.2.1 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed int >::type_for_add

Definition at line 855 of file checked_int.inlines.hh.

11.122.2.2 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed int >::type_for_mul

Definition at line 857 of file checked_int.inlines.hh.

11.122.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed int >::type_for_neg

Definition at line 854 of file checked_int.inlines.hh.

11.122.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed int >::type_for_sub

Definition at line 856 of file checked_int.inlines.hh.

11.122.3 Member Function Documentation

11.122.3.1 Parma_Polyhedra_Library::Checked::Larger< signed int >::const_bool_nodef (use_for_mul, true)

11.122.3.2 Parma_Polyhedra_Library::Checked::Larger< signed int >::const_bool_nodef (use_for_sub, (LONG_BITS==64))

11.122.3.3 Parma_Polyhedra_Library::Checked::Larger< signed int >::const_bool_nodef (use_for_add, (LONG_BITS==64))

11.122.3.4 Parma_Polyhedra_Library::Checked::Larger< signed int >::const_bool_nodef (use_for_neg, (LONG_BITS==64))

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.123 Parma_Polyhedra_Library::Checked::Larger< signed long > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef int_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef int_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, false)
- const_bool_nodef (use_for_add, false)

- const_bool_nodef (use_for_sub, false)
- const_bool_nodef (use_for_mul,(LONG_BITS==32))

11.123.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< signed long >

Definition at line 873 of file checked_int.inlines.hh.

11.123.2 Member Typedef Documentation

11.123.2.1 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long >::type_for_add

Definition at line 879 of file checked_int.inlines.hh.

11.123.2.2 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long >::type_for_mul

Definition at line 881 of file checked_int.inlines.hh.

11.123.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long >::type_for_neg

Definition at line 878 of file checked_int.inlines.hh.

11.123.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long >::type_for_sub

Definition at line 880 of file checked_int.inlines.hh.

11.123.3 Member Function Documentation

11.123.3.1 Parma_Polyhedra_Library::Checked::Larger< signed long >::const_bool_nodef (use_for_mul, (LONG_BITS==32))

11.123.3.2 Parma_Polyhedra_Library::Checked::Larger< signed long >::const_bool_nodef (use_for_sub, false)

11.123.3.3 Parma_Polyhedra_Library::Checked::Larger< signed long >::const_bool_nodef (use_for_add, false)

11.123.3.4 Parma_Polyhedra_Library::Checked::Larger< signed long >::const_bool_nodef (use_for_neg, false)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.124 Parma_Polyhedra_Library::Checked::Larger< signed long long > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef int_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef int_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, false)
- const_bool_nodef (use_for_add, false)
- const_bool_nodef (use_for_sub, false)
- const_bool_nodef (use_for_mul, false)

11.124.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< signed long long >

Definition at line 897 of file checked_int.inlines.hh.

11.124.2 Member Typedef Documentation

11.124.2.1 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long long >::type_for_add

Definition at line 903 of file checked_int.inlines.hh.

11.124.2.2 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long long >::type_for_mul

Definition at line 905 of file checked_int.inlines.hh.

11.124.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long long >::type_for_neg

Definition at line 902 of file checked_int.inlines.hh.

11.124.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< signed long long >::type_for_sub

Definition at line 904 of file checked_int.inlines.hh.

11.124.3 Member Function Documentation

11.124.3.1 Parma_Polyhedra_Library::Checked::Larger< signed long long >::const_bool_nodef (use_for_mul, false)

11.124.3.2 Parma_Polyhedra_Library::Checked::Larger< signed long long >::const_bool_nodef (use_for_sub, false)

11.124.3.3 Parma_Polyhedra_Library::Checked::Larger< signed long long >::const_bool_nodef (use_for_add, false)

11.124.3.4 Parma_Polyhedra_Library::Checked::Larger< signed long long >::const_bool_nodef (use_for_neg, false)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.125 Parma_Polyhedra_Library::Checked::Larger< signed short > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast32_t type_for_neg
- typedef int_fast32_t type_for_add
- typedef int_fast32_t type_for_sub
- typedef int_fast32_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, true)
- const_bool_nodef (use_for_add, true)
- const_bool_nodef (use_for_sub, true)
- const_bool_nodef (use_for_mul, true)

11.125.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< signed short >

Definition at line 825 of file checked_int.inlines.hh.

11.125.2 Member Typedef Documentation

11.125.2.1 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< signed short >::type_for_add

Definition at line 831 of file checked_int.inlines.hh.

11.125.2.2 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< signed short >::type_for_mul

Definition at line 833 of file checked_int.inlines.hh.

11.125.2.3 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< signed short >::type_for_neg

Definition at line 830 of file checked_int.inlines.hh.

11.125.2.4 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< signed short >::type_for_sub

Definition at line 832 of file checked_int.inlines.hh.

11.125.3 Member Function Documentation

11.125.3.1 Parma_Polyhedra_Library::Checked::Larger< signed short >::const_bool_nodef (use_for_mul, true)

11.125.3.2 Parma_Polyhedra_Library::Checked::Larger< signed short >::const_bool_nodef (use_for_sub, true)

11.125.3.3 Parma_Polyhedra_Library::Checked::Larger< signed short >::const_bool_nodef (use_for_add, true)

11.125.3.4 Parma_Polyhedra_Library::Checked::Larger< signed short >::const_bool_nodef (use_for_neg, true)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.126 Parma_Polyhedra_Library::Checked::Larger< unsigned char > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast16_t type_for_neg
- typedef uint_fast16_t type_for_add
- typedef int_fast16_t type_for_sub
- typedef uint_fast16_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, true)
- const_bool_nodef (use_for_add, true)
- const_bool_nodef (use_for_sub, true)
- const_bool_nodef (use_for_mul, true)

11.126.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< unsigned char >

Definition at line 813 of file checked_int.inlines.hh.

11.126.2 Member Typedef Documentation

11.126.2.1 typedef uint_fast16_t Parma_Polyhedra_Library::Checked::Larger< unsigned char >::type_for_add

Definition at line 819 of file checked_int.inlines.hh.

11.126.2.2 typedef uint_fast16_t Parma_Polyhedra_Library::Checked::Larger< unsigned char >::type_for_mul

Definition at line 821 of file checked_int.inlines.hh.

11.126.2.3 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< unsigned char >::type_for_neg

Definition at line 818 of file checked_int.inlines.hh.

11.126.2.4 typedef int_fast16_t Parma_Polyhedra_Library::Checked::Larger< unsigned char >::type_for_sub

Definition at line 820 of file checked_int.inlines.hh.

11.126.3 Member Function Documentation

11.126.3.1 Parma_Polyhedra_Library::Checked::Larger< unsigned char >::const_bool_nodef (use_for_mul, true)

11.126.3.2 Parma_Polyhedra_Library::Checked::Larger< unsigned char >::const_bool_nodef (use_for_sub, true)

11.126.3.3 Parma_Polyhedra_Library::Checked::Larger< unsigned char >::const_bool_nodef (use_for_add, true)

11.126.3.4 Parma_Polyhedra_Library::Checked::Larger< unsigned char >::const_bool_nodef (use_for_neg, true)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.127 Parma_Polyhedra_Library::Checked::Larger< unsigned int > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef uint_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef uint_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg,(LONG_BITS==64))
- const_bool_nodef (use_for_add,(LONG_BITS==64))
- const_bool_nodef (use_for_sub,(LONG_BITS==64))
- const_bool_nodef (use_for_mul, true)

11.127.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< unsigned int >

Definition at line 861 of file checked_int.inlines.hh.

11.127.2 Member Typedef Documentation

11.127.2.1 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned int >::type_for_add

Definition at line 867 of file checked_int.inlines.hh.

11.127.2.2 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned int >::type_for_mul

Definition at line 869 of file checked_int.inlines.hh.

11.127.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned int >::type_for_neg

Definition at line 866 of file checked_int.inlines.hh.

11.127.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned int >::type_for_sub

Definition at line 868 of file checked_int.inlines.hh.

11.127.3 Member Function Documentation

11.127.3.1 Parma_Polyhedra_Library::Checked::Larger< unsigned int >::const_bool_nodef (use_for_mul, true)

11.127.3.2 Parma_Polyhedra_Library::Checked::Larger< unsigned int >::const_bool_nodef (use_for_sub, (LONG_BITS==64))

11.127.3.3 Parma_Polyhedra_Library::Checked::Larger< unsigned int >::const_bool_nodef (use_for_add, (LONG_BITS==64))

11.127.3.4 Parma_Polyhedra_Library::Checked::Larger< unsigned int >::const_bool_nodef (use_for_neg, (LONG_BITS==64))

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.128 Parma_Polyhedra_Library::Checked::Larger< unsigned long > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef uint_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef uint_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, false)
- const_bool_nodef (use_for_add, false)

- const_bool_nodef (use_for_sub, false)
- const_bool_nodef (use_for_mul,(LONG_BITS==32))

11.128.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< unsigned long >

Definition at line 885 of file checked_int.inlines.hh.

11.128.2 Member Typedef Documentation

11.128.2.1 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long >::type_for_add

Definition at line 891 of file checked_int.inlines.hh.

11.128.2.2 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long >::type_for_mul

Definition at line 893 of file checked_int.inlines.hh.

11.128.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long >::type_for_neg

Definition at line 890 of file checked_int.inlines.hh.

11.128.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long >::type_for_sub

Definition at line 892 of file checked_int.inlines.hh.

11.128.3 Member Function Documentation

11.128.3.1 Parma_Polyhedra_Library::Checked::Larger< unsigned long >::const_bool_nodef (use_for_mul, (LONG_BITS==32))

11.128.3.2 Parma_Polyhedra_Library::Checked::Larger< unsigned long >::const_bool_nodef (use_for_sub, false)

11.128.3.3 Parma_Polyhedra_Library::Checked::Larger< unsigned long >::const_bool_nodef (use_for_add, false)

11.128.3.4 Parma_Polyhedra_Library::Checked::Larger< unsigned long >::const_bool_nodef (use_for_neg, false)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.129 Parma_Polyhedra_Library::Checked::Larger< unsigned long long > Struct Template Reference

`#include <checked_int.inlines.hh>`

Public Types

- typedef int_fast64_t type_for_neg
- typedef uint_fast64_t type_for_add
- typedef int_fast64_t type_for_sub
- typedef uint_fast64_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, false)
- const_bool_nodef (use_for_add, false)
- const_bool_nodef (use_for_sub, false)
- const_bool_nodef (use_for_mul, false)

11.129.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< unsigned long long >

Definition at line 909 of file checked_int.inlines.hh.

11.129.2 Member Typedef Documentation

11.129.2.1 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::type_for_add

Definition at line 915 of file checked_int.inlines.hh.

11.129.2.2 typedef uint_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::type_for_mul

Definition at line 917 of file checked_int.inlines.hh.

11.129.2.3 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::type_for_neg

Definition at line 914 of file checked_int.inlines.hh.

11.129.2.4 typedef int_fast64_t Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::type_for_sub

Definition at line 916 of file checked_int.inlines.hh.

11.129.3 Member Function Documentation

11.129.3.1 Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::const_bool_nodef (use_for_mul, false)

11.129.3.2 Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::const_bool_nodef (use_for_sub, false)

11.129.3.3 Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::const_bool_nodef (use_for_add, false)

11.129.3.4 Parma_Polyhedra_Library::Checked::Larger< unsigned long long >::const_bool_nodef (use_for_neg, false)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.130 Parma_Polyhedra_Library::Checked::Larger< unsigned short > Struct Template Reference

```
#include <checked_int.inlines.hh>
```

Public Types

- typedef int_fast32_t type_for_neg
- typedef uint_fast32_t type_for_add
- typedef int_fast32_t type_for_sub
- typedef uint_fast32_t type_for_mul

Public Member Functions

- const_bool_nodef (use_for_neg, true)
- const_bool_nodef (use_for_add, true)
- const_bool_nodef (use_for_sub, true)
- const_bool_nodef (use_for_mul, true)

11.130.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Checked::Larger< unsigned short >

Definition at line 837 of file checked_int.inlines.hh.

11.130.2 Member Typedef Documentation

11.130.2.1 typedef uint_fast32_t Parma_Polyhedra_Library::Checked::Larger< unsigned short >::type_for_add

Definition at line 843 of file checked_int.inlines.hh.

11.130.2.2 typedef uint_fast32_t Parma_Polyhedra_Library::Checked::Larger< unsigned short >::type_for_mul

Definition at line 845 of file checked_int.inlines.hh.

11.130.2.3 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< unsigned short >::type_for_neg

Definition at line 842 of file checked_int.inlines.hh.

11.130.2.4 typedef int_fast32_t Parma_Polyhedra_Library::Checked::Larger< unsigned short >::type_for_sub

Definition at line 844 of file checked_int.inlines.hh.

11.130.3 Member Function Documentation

11.130.3.1 Parma_Polyhedra_Library::Checked::Larger< unsigned short >::const_bool_nodef (use_for_mul, true)

11.130.3.2 Parma_Polyhedra_Library::Checked::Larger< unsigned short >::const_bool_nodef (use_for_sub, true)

11.130.3.3 Parma_Polyhedra_Library::Checked::Larger< unsigned short >::const_bool_nodef (use_for_add, true)

11.130.3.4 Parma_Polyhedra_Library::Checked::Larger< unsigned short >::const_bool_nodef (use_for_neg, true)

The documentation for this struct was generated from the following file:

- checked_int.inlines.hh

11.131 Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS > Class Template Reference

Wraps a limited widening method into a function object.

`#include <Widening_Function.defs.hh>`

Public Types

- typedef void(PSET::* Limited_Widening_Method)(const PSET &, const CSYS &, unsigned *)

  The (parametric) type of a limited widening method.

Public Member Functions

- Limited_Widening_Function (Limited_Widening_Method lwm, const CSYS &cs)

  Constructor.

- void operator() (PSET &x, const PSET &y, unsigned *tp=0) const

  Function-application operator.

Private Attributes

- Limited_Widening_Method lw_method
  The limited widening method.
- const CSYS & limiting_cs
  A constant reference to the constraint system limiting the widening.

11.131.1 Detailed Description

template<typename PSET, typename CSYS> class Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >

Wraps a limited widening method into a function object.

Definition at line 62 of file Widening_Function.defs.hh.

11.131.2 Member Typedef Documentation

11.131.2.1 template<typename PSET, typename CSYS> typedef void(PSET::* Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::Limited_Widening_Method)(const PSET &, const CSYS &, unsigned *)

The (parametric) type of a limited widening method.

Definition at line 65 of file Widening_Function.defs.hh.

11.131.3 Constructor & Destructor Documentation

11.131.3.1 template<typename PSET , typename CSYS > Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::Limited_Widening_Function (Limited_Widening_Method *lwm*, const CSYS & *cs*) [inline]

Constructor.

Parameters

lwm The limited widening method.

cs The constraint system limiting the widening.

Definition at line 45 of file Widening_Function.inlines.hh.

11.131.4 Member Function Documentation

11.131.4.1 template<typename PSET , typename CSYS > void Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::operator() (PSET & *x*, const PSET & *y*, unsigned * *tp* = 0) const [inline]

Function-application operator.

Computes `(x.*lwm)(y, cs, tp)`, where `lwm` and `cs` are the limited widening method and the constraint system stored at construction time.

Definition at line 53 of file Widening_Function.inlines.hh.

References Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::limiting_cs, and Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::lw_method.

11.131.5 Member Data Documentation

11.131.5.1 template<typename PSET, typename CSYS> const CSYS& Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::limiting_cs [private]

A constant reference to the constraint system limiting the widening.

Definition at line 92 of file Widening_Function.defs.hh.

Referenced by Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::operator()().

11.131.5.2 template<typename PSET, typename CSYS> Limited_Widening_Method Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::lw_method [private]

The limited widening method.

Definition at line 90 of file Widening_Function.defs.hh.

Referenced by Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >::operator()().

The documentation for this class was generated from the following files:

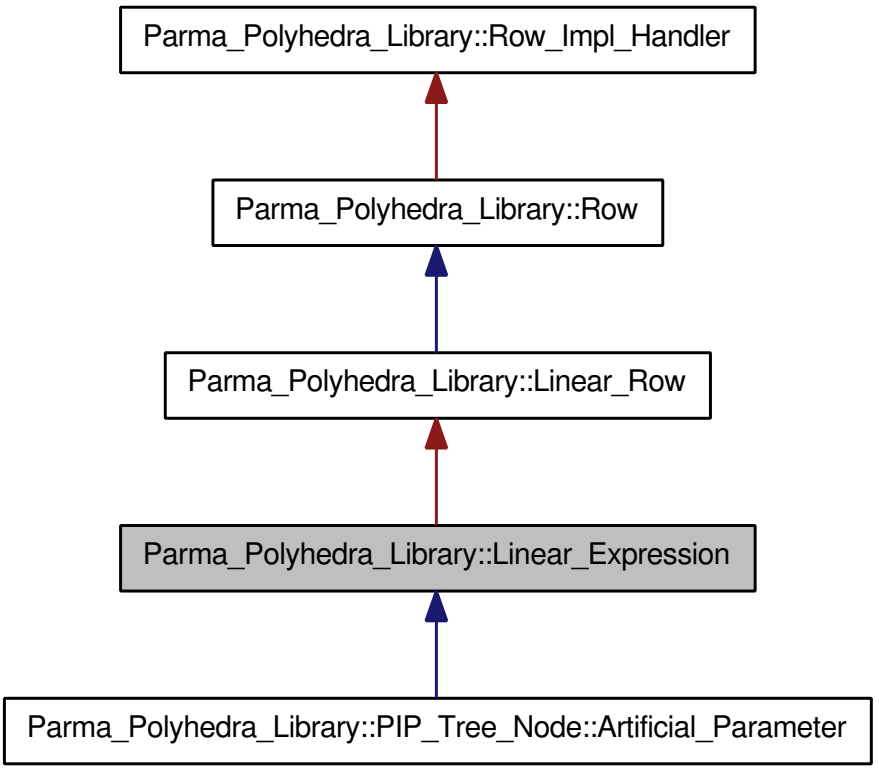
- Widening_Function.defs.hh
- Widening_Function.inlines.hh

11.132 Parma_Polyhedra_Library::Linear_Expression Class Reference

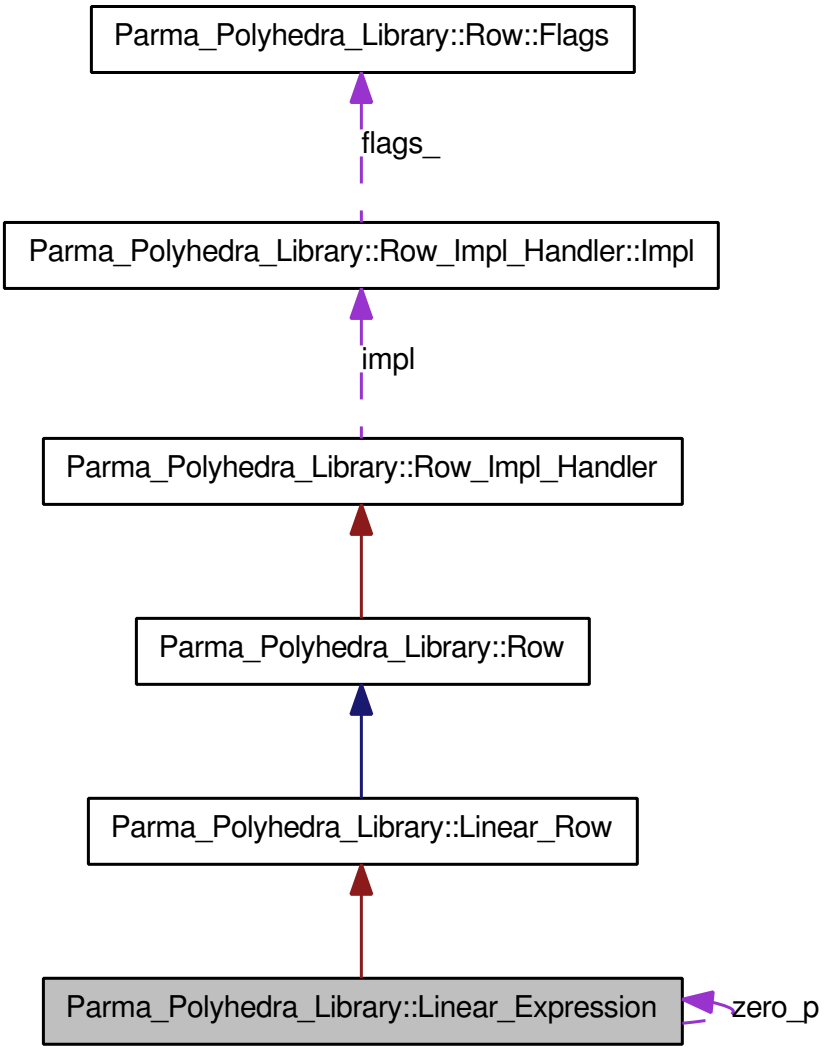
A linear expression.

`#include <Linear_Expression.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Linear_Expression:



Collaboration diagram for Parma_Polyhedra_Library::Linear_Expression:



Public Member Functions

- Linear_Expression ()

  Default constructor: returns a copy of Linear_Expression::zero().

- Linear_Expression (const Linear_Expression &e)

  Ordinary copy constructor.

- ~Linear_Expression ()

  Destructor.

- Linear_Expression (Coefficient_traits::const_reference n)

  Builds the linear expression corresponding to the inhomogeneous term `n`.

- Linear_Expression (Variable v)

Builds the linear expression corresponding to the variable `v`.

- Linear_Expression (const Constraint &c)

  Builds the linear expression corresponding to constraint `c`.

- Linear_Expression (const Generator &g)

  Builds the linear expression corresponding to generator `g` *(for points and closure points, the divisor is not copied).*

- Linear_Expression (const Grid_Generator &g)

  Builds the linear expression corresponding to grid generator `g` *(for points, parameters and lines the divisor is not copied).*

- Linear_Expression (const Congruence &cg)

  Builds the linear expression corresponding to congruence `cg`.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Coefficient_traits::const_reference coefficient (Variable v) const

  Returns the coefficient of `v` *in* `*this`.

- Coefficient_traits::const_reference inhomogeneous_term () const

  Returns the inhomogeneous term of `*this`.

- bool is_zero () const

  Returns `true` *if and only if* `*this` *is* 0.

- bool all_homogeneous_terms_are_zero () const

  Returns `true` *if and only if all the homogeneous terms of* `*this` *are* 0.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void swap (Linear_Expression &y)

  Swaps `*this` *with* `y`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Linear_Expression can handle.

- static void initialize ()

  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static const Linear_Expression & zero ()

  Returns the (zero-dimension space) constant 0.

Private Member Functions

- Linear_Expression (const Linear_Expression &e, dimension_type sz)

  Copy constructor with a specified space dimension.

- Linear_Expression (dimension_type sz, bool)

  Implementation sizing constructor.

- Linear_Expression (Variable v, Variable w)

  Builds the linear expression corresponding to the difference of `v` *and* `w`.

Static Private Attributes

- static const Linear_Expression * zero_p = 0

  Holds (between class initialization and finalization) a pointer to the (zero-dimension space) constant 0.

Friends

- class Parma_Polyhedra_Library::Scalar_Products
- class Parma_Polyhedra_Library::Constraint
- class Parma_Polyhedra_Library::Generator
- class Parma_Polyhedra_Library::Grid_Generator
- class Parma_Polyhedra_Library::Congruence
- class Parma_Polyhedra_Library::Polyhedron
- class Parma_Polyhedra_Library::Grid

- class Parma_Polyhedra_Library::Constraint_System
- class Parma_Polyhedra_Library::Generator_System
- class Parma_Polyhedra_Library::Congruence_System
- class Parma_Polyhedra_Library::Grid_Generator_System
- Linear_Expression operator+ (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the linear expression `e1` + `e2`.

- Linear_Expression operator+ (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the linear expression `n` + `e`.

- Linear_Expression operator+ (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `e` + `n`.

- Linear_Expression operator+ (Variable v, const Linear_Expression &e)

  Returns the linear expression `v` + `e`.

- Linear_Expression operator+ (Variable v, Variable w)

  Returns the linear expression `v` + `w`.

- Linear_Expression operator- (const Linear_Expression &e)

  Returns the linear expression - `e`.

- Linear_Expression operator- (const Linear_Expression &e1, const Linear_Expression &e2)

  Returns the linear expression `e1` - `e2`.

- Linear_Expression operator- (Variable v, Variable w)

  Returns the linear expression `v` - `w`.

- Linear_Expression operator- (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the linear expression `n` - `e`.

- Linear_Expression operator- (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `e` - `n`.

- Linear_Expression operator- (Variable v, const Linear_Expression &e)

  Returns the linear expression `v` - `e`.

- Linear_Expression operator- (const Linear_Expression &e, Variable v)

  Returns the linear expression `e` - `v`.

- Linear_Expression operator∗ (Coefficient_traits::const_reference n, const Linear_Expression &e)

  Returns the linear expression `n` ∗ `e`.

- Linear_Expression operator∗ (const Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `e` ∗ `n`.

- Linear_Expression & operator+= (Linear_Expression &e1, const Linear_Expression &e2)

  Returns the linear expression `e1` + `e2` *and assigns it to* `e1`.

- Linear_Expression & operator+= (Linear_Expression &e, Variable v)

Returns the linear expression `e + v` *and assigns it to* `e`.

- Linear_Expression & operator+= (Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `e + n` *and assigns it to* `e`.

- Linear_Expression & operator-= (Linear_Expression &e1, const Linear_Expression &e2)

  Returns the linear expression `e1 - e2` *and assigns it to* `e1`.

- Linear_Expression & operator-= (Linear_Expression &e, Variable v)

  Returns the linear expression `e - v` *and assigns it to* `e`.

- Linear_Expression & operator-= (Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `e - n` *and assigns it to* `e`.

- Linear_Expression & operator∗= (Linear_Expression &e, Coefficient_traits::const_reference n)

  Returns the linear expression `n * e` *and assigns it to* `e`.

- Linear_Expression & add_mul_assign (Linear_Expression &e, Coefficient_traits::const_reference n, Variable v)

  Returns the linear expression `e + n * v` *and assigns it to* `e`.

- Linear_Expression & sub_mul_assign (Linear_Expression &e, Coefficient_traits::const_reference n, Variable v)

  Returns the linear expression `e - n * v` *and assigns it to* `e`.

- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Linear_Expression &e)

Related Functions

(Note that these are not member functions.)

- Linear_Expression operator+ (const Linear_Expression &e, Variable v)

  Returns the linear expression `e + v`.

- Linear_Expression operator+ (const Linear_Expression &e)

  Returns the linear expression `e`.

- std::ostream & operator<< (std::ostream &s, const Linear_Expression &e)

  Output operator.

- void swap (Parma_Polyhedra_Library::Linear_Expression &x, Parma_Polyhedra_Library::Linear_-Expression &y)

  Specializes `std::swap`.

11.132.1 Detailed Description

A linear expression. An object of the class Linear_Expression represents the linear expression

$$\sum_{i=0}^{n-1} a_i x_i + b$$

where n is the dimension of the vector space, each a_i is the integer coefficient of the i-th variable x_i and b is the integer for the inhomogeneous term.

How to build a linear expression.

Linear expressions are the basic blocks for defining both constraints (i.e., linear equalities or inequalities) and generators (i.e., lines, rays, points and closure points). A full set of functions is defined to provide a convenient interface for building complex linear expressions starting from simpler ones and from objects of the classes Variable and Coefficient: available operators include unary negation, binary addition and subtraction, as well as multiplication by a Coefficient. The space dimension of a linear expression is defined as the maximum space dimension of the arguments used to build it: in particular, the space dimension of a Variable `x` is defined as `x.id()+1`, whereas all the objects of the class Coefficient have space dimension zero.

Example

The following code builds the linear expression $4x - 2y - z + 14$, having space dimension 3:

```
Linear_Expression e = 4*x - 2*y - z + 14;
```

Another way to build the same linear expression is:

```
Linear_Expression e1 = 4*x;
Linear_Expression e2 = 2*y;
Linear_Expression e3 = z;
Linear_Expression e = Linear_Expression(14);
e += e1 - e2 - e3;
```

Note that `e1`, `e2` and `e3` have space dimension 1, 2 and 3, respectively; also, in the fourth line of code, `e` is created with space dimension zero and then extended to space dimension 3 in the fifth line.

Definition at line 249 of file Linear_Expression.defs.hh.

11.132.2 Constructor & Destructor Documentation

11.132.2.1 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression () [inline]

Default constructor: returns a copy of Linear_Expression::zero().

Definition at line 39 of file Linear_Expression.inlines.hh.

Referenced by initialize().

11.132.2.2 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Linear_Expression & *e*) [inline]

Ordinary copy constructor.

Definition at line 49 of file Linear_Expression.inlines.hh.

11.132.2.3 Parma_Polyhedra_Library::Linear_Expression::~Linear_Expression () [inline]

Destructor.

Definition at line 54 of file Linear_Expression.inlines.hh.

11.132.2.4 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (Coefficient_traits::const_reference *n*) [inline, explicit]

Builds the linear expression corresponding to the inhomogeneous term `n`.

Definition at line 64 of file Linear_Expression.inlines.hh.

11.132.2.5 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (Variable *v*)

Builds the linear expression corresponding to the variable `v`.

Exceptions

std::length_error Thrown if the space dimension of `v` exceeds `Linear_Expression::max_space_dimension()`.

Definition at line 82 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Variable::space_dimension().

11.132.2.6 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Constraint & *c*) [explicit]

Builds the linear expression corresponding to constraint `c`.

Given the constraint $c = \left(\sum_{i=0}^{n-1} a_i x_i + b \bowtie 0\right)$, where $\bowtie \in \{=, \geq, >\}$, this builds the linear expression $\sum_{i=0}^{n-1} a_i x_i + b$. If `c` is an inequality (resp., equality) constraint, then the built linear expression is unique up to a positive (resp., non-zero) factor.

Definition at line 37 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Row::size().

11.132.2.7 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Generator & *g*) [explicit]

Builds the linear expression corresponding to generator g (for points and closure points, the divisor is not copied).

Given the generator $g = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$ (where, for lines and rays, we have $d = 1$), this builds the linear expression $\sum_{i=0}^{n-1} a_i x_i$. The inhomogeneous term of the linear expression will always be 0. If g is a ray, point or closure point (resp., a line), then the linear expression is unique up to a positive (resp., non-zero) factor.

Definition at line 44 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Row::size().

11.132.2.8 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Grid_Generator & *g*) [explicit]

Builds the linear expression corresponding to grid generator g (for points, parameters and lines the divisor is not copied).

Given the grid generator $g = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$ this builds the linear expression $\sum_{i=0}^{n-1} a_i x_i$. The inhomogeneous term of the linear expression is always 0.

Definition at line 52 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Row::size().

11.132.2.9 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Congruence & *cg*) [explicit]

Builds the linear expression corresponding to congruence cg.

Given the congruence $cg = \left(\sum_{i=0}^{n-1} a_i x_i + b = 0 \pmod{m}\right)$, this builds the linear expression $\sum_{i=0}^{n-1} a_i x_i + b$.

Definition at line 75 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Row::size().

11.132.2.10 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (const Linear_Expression & *e*, dimension_type *sz*) [inline, private]

Copy constructor with a specified space dimension.

Definition at line 58 of file Linear_Expression.inlines.hh.

11.132.2.11 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (dimension_type *sz*, bool) [inline, private]

Implementation sizing constructor.

The bool parameter is just to avoid problems with the constructor Linear_Expression(Coefficient_traits::const_reference n).

Definition at line 44 of file Linear_Expression.inlines.hh.

11.132.2.12 Parma_Polyhedra_Library::Linear_Expression::Linear_Expression (Variable *v*, Variable *w*) [private]

Builds the linear expression corresponding to the difference of `v` and `w`.

Exceptions

std::length_error Thrown if the space dimension of `v` or the one of `w` exceed `Linear_Expression::max_space_dimension()`.

Definition at line 94 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Linear_Row::construct(), max_space_dimension(), and Parma_Polyhedra_Library::Variable::space_dimension().

11.132.3 Member Function Documentation

11.132.3.1 bool Parma_Polyhedra_Library::Linear_Expression::all_homogeneous_terms_are_zero () const [inline]

Returns `true` if and only if all the homogeneous terms of `*this` are 0.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 92 of file Linear_Expression.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), and Parma_Polyhedra_Library::Grid::Grid().

11.132.3.2 void Parma_Polyhedra_Library::Linear_Expression::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 168 of file Linear_Expression.inlines.hh.

References ascii_dump().

11.132.3.3 void Parma_Polyhedra_Library::Linear_Expression::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Referenced by Parma_Polyhedra_Library::MIP_Problem::ascii_dump(), and ascii_dump().

11.132.3.4 bool Parma_Polyhedra_Library::Linear_Expression::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 173 of file Linear_Expression.inlines.hh.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::ascii_load(), and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::ascii_load().

11.132.3.5 Coefficient_traits::const_reference Parma_Polyhedra_Library::Linear_Expression::coefficient (Variable *v*) const [inline]

Returns the coefficient of `v` in `*this`.

Definition at line 75 of file Linear_Expression.inlines.hh.

References Parma_Polyhedra_Library::Coefficient_zero(), Parma_Polyhedra_Library::Variable::id(), space_dimension(), and Parma_Polyhedra_Library::Variable::space_dimension().

Referenced by Parma_Polyhedra_Library::Polyhedron::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Polyhedron::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_u_minus_v_bounds(), Parma_Polyhedra_Library::BD_Shape< T >::deduce_v_minus_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(),

Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::operator==(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.132.3.6 memory_size_type Parma_Polyhedra_Library::Linear_Expression::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 103 of file Linear_Expression.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::external_memory_in_bytes().

11.132.3.7 void Parma_Polyhedra_Library::Linear_Expression::finalize () [static]

Finalizes the class.

Definition at line 69 of file Linear_Expression.cc.

References zero_p.

11.132.3.8 Coefficient_traits::const_reference Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term () const [inline]

Returns the inhomogeneous term of `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 82 of file Linear_Expression.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Octagonal_Shape<

T >::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::operator==(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.132.3.9 void Parma_Polyhedra_Library::Linear_Expression::initialize () [static]

Initializes the class.

Definition at line 63 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Coefficient_zero(), Linear_Expression(), and zero_p.

11.132.3.10 bool Parma_Polyhedra_Library::Linear_Expression::is_zero () const [inline]

Returns `true` if and only if `*this` is 0.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 87 of file Linear_Expression.inlines.hh.

11.132.3.11 dimension_type Parma_Polyhedra_Library::Linear_Expression::max_space_dimension () [inline, static]

Returns the maximum space dimension a Linear_Expression can handle.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 34 of file Linear_Expression.inlines.hh.

Referenced by Linear_Expression().

11.132.3.12 bool Parma_Polyhedra_Library::Linear_Expression::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 397 of file Linear_Expression.cc.

Referenced by Parma_Polyhedra_Library::MIP_Problem::OK().

11.132.3.13 void Parma_Polyhedra_Library::Linear_Expression::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

11.132.3.14 dimension_type Parma_Polyhedra_Library::Linear_Expression::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 70 of file Linear_Expression.inlines.hh.

References Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Grid::affine_image(), Parma_Polyhedra_Library::Grid_Generator_System::affine_image(), Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Polyhedron::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Grid::affine_preimage(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Grid::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Grid::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Polyhedron::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Grid::bounds(), Parma_Polyhedra_Library::Box< ITV >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Grid::bounds_no_check(), coefficient(), Parma_Polyhedra_Library::Congruence::create(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Grid::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_-

min(), Parma_Polyhedra_Library::Box< ITV >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::MIP_Problem::MIP_Problem(), Parma_Polyhedra_Library::MIP_Problem::OK(), operator<<(), Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::operator==(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), Parma_Polyhedra_Library::MIP_Problem::set_objective_function(), Parma_Polyhedra_Library::Grid::simplify_using_context_assign(), Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), Parma_Polyhedra_Library::Box< ITV >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::BD_Shape< T >::throw_dimension_incompatible().

11.132.3.15 void Parma_Polyhedra_Library::Linear_Expression::swap (Linear_Expression & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 163 of file Linear_Expression.inlines.hh.

Referenced by swap().

11.132.3.16 memory_size_type Parma_Polyhedra_Library::Linear_Expression::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 108 of file Linear_Expression.inlines.hh.

11.132.3.17 const Linear_Expression & Parma_Polyhedra_Library::Linear_Expression::zero () [inline, static]

Returns the (zero-dimension space) constant 0.

Definition at line 97 of file Linear_Expression.inlines.hh.

References zero_p.

Referenced by Parma_Polyhedra_Library::Constraint::initialize(), and Parma_Polyhedra_Library::Congruence::initialize().

11.132.4 Friends And Related Function Documentation

11.132.4.1 PPL::Linear_Expression & add_mul_assign (Linear_Expression & *e*, Coefficient_traits::const_reference *n*, Variable *v*) [friend]

Returns the linear expression `e + n * v` and assigns it to `e`.

Definition at line 360 of file Linear_Expression.cc.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter().

11.132.4.2 Linear_Expression operator* (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `e * n`.

Definition at line 144 of file Linear_Expression.inlines.hh.

11.132.4.3 PPL::Linear_Expression operator* (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the linear expression `n * e`.

Definition at line 273 of file Linear_Expression.cc.

11.132.4.4 PPL::Linear_Expression & operator*= (Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `n * e` and assigns it to `e`.

Definition at line 351 of file Linear_Expression.cc.

11.132.4.5 Linear_Expression operator+ (const Linear_Expression & *e*) [related]

Returns the linear expression `e`.

Definition at line 114 of file Linear_Expression.inlines.hh.

11.132.4.6 Linear_Expression operator+ (const Linear_Expression & *e*, Variable *v*) [related]

Returns the linear expression `e + v`.

Definition at line 126 of file Linear_Expression.inlines.hh.

11.132.4.7 PPL::Linear_Expression operator+ (Variable *v*, Variable *w*) [friend]

Returns the linear expression `v + w`.

Definition at line 170 of file Linear_Expression.cc.

11.132.4.8 PPL::Linear_Expression operator+ (Variable *v*, const Linear_Expression & *e*) [friend]

Returns the linear expression `v` + `e`.

Definition at line 146 of file Linear_Expression.cc.

11.132.4.9 Linear_Expression operator+ (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `e` + `n`.

Definition at line 120 of file Linear_Expression.inlines.hh.

11.132.4.10 PPL::Linear_Expression operator+ (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the linear expression `n` + `e`.

Definition at line 161 of file Linear_Expression.cc.

11.132.4.11 PPL::Linear_Expression operator+ (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the linear expression `e1` + `e2`.

Definition at line 113 of file Linear_Expression.cc.

11.132.4.12 Linear_Expression & operator+= (Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `e` + `n` and assigns it to `e`.

Definition at line 150 of file Linear_Expression.inlines.hh.

11.132.4.13 PPL::Linear_Expression & operator+= (Linear_Expression & *e*, Variable *v*) [friend]

Returns the linear expression `e` + `v` and assigns it to `e`.

Exceptions

std::length_error Thrown if the space dimension of `v` exceeds `Linear_Expression::max_space_dimension()`.

Definition at line 300 of file Linear_Expression.cc.

11.132.4.14 PPL::Linear_Expression & operator+= (Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the linear expression `e1 + e2` and assigns it to `e1`.

Definition at line 283 of file Linear_Expression.cc.

11.132.4.15 PPL::Linear_Expression operator- (const Linear_Expression & *e*, Variable *v*) [friend]

Returns the linear expression `e - v`.

Definition at line 247 of file Linear_Expression.cc.

11.132.4.16 PPL::Linear_Expression operator- (Variable *v*, const Linear_Expression & *e*) [friend]

Returns the linear expression `v - e`.

Definition at line 229 of file Linear_Expression.cc.

11.132.4.17 Linear_Expression operator- (const Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `e - n`.

Definition at line 132 of file Linear_Expression.inlines.hh.

11.132.4.18 PPL::Linear_Expression operator- (Coefficient_traits::const_reference *n*, const Linear_Expression & *e*) [friend]

Returns the linear expression `n - e`.

Definition at line 262 of file Linear_Expression.cc.

11.132.4.19 Linear_Expression operator- (Variable *v*, Variable *w*) [friend]

Returns the linear expression `v - w`.

Definition at line 138 of file Linear_Expression.inlines.hh.

11.132.4.20 PPL::Linear_Expression operator- (const Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the linear expression `e1 - e2`.

Definition at line 196 of file Linear_Expression.cc.

11.132.4.21 PPL::Linear_Expression operator- (const Linear_Expression & *e*) [friend]

Returns the linear expression - `e`.

Definition at line 187 of file Linear_Expression.cc.

11.132.4.22 Linear_Expression & operator-= (Linear_Expression & *e*, Coefficient_traits::const_reference *n*) [friend]

Returns the linear expression `e - n` and assigns it to `e`.

Definition at line 157 of file Linear_Expression.inlines.hh.

11.132.4.23 PPL::Linear_Expression & operator-= (Linear_Expression & *e*, Variable *v*) [friend]

Returns the linear expression `e - v` and assigns it to `e`.

Exceptions

std::length_error Thrown if the space dimension of `v` exceeds `Linear_Expression::max_space_dimension()`.

Definition at line 334 of file Linear_Expression.cc.

11.132.4.24 PPL::Linear_Expression & operator-= (Linear_Expression & *e1*, const Linear_Expression & *e2*) [friend]

Returns the linear expression `e1 - e2` and assigns it to `e1`.

Definition at line 317 of file Linear_Expression.cc.

11.132.4.25 std::ostream & operator<< (std::ostream & *s*, const Linear_Expression & *e*) [related]

Output operator.

Definition at line 403 of file Linear_Expression.cc.

References Parma_Polyhedra_Library::Coefficient_zero(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and space_dimension().

11.132.4.26 friend class Parma_Polyhedra_Library::Congruence [friend]

Definition at line 386 of file Linear_Expression.defs.hh.

11.132.4.27 friend class Parma_Polyhedra_Library::Congruence_System [friend]

Definition at line 401 of file Linear_Expression.defs.hh.

11.132.4.28 friend class Parma_Polyhedra_Library::Constraint [friend]

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 382 of file Linear_Expression.defs.hh.

11.132.4.29 friend class Parma_Polyhedra_Library::Constraint_System [friend]

Definition at line 393 of file Linear_Expression.defs.hh.

11.132.4.30 friend class Parma_Polyhedra_Library::Generator [friend]

Reimplemented from Parma_Polyhedra_Library::Linear_Row.

Definition at line 383 of file Linear_Expression.defs.hh.

11.132.4.31 friend class Parma_Polyhedra_Library::Generator_System [friend]

Definition at line 397 of file Linear_Expression.defs.hh.

11.132.4.32 friend class Parma_Polyhedra_Library::Grid [friend]

Definition at line 389 of file Linear_Expression.defs.hh.

11.132.4.33 friend class Parma_Polyhedra_Library::Grid_Generator [friend]

Definition at line 385 of file Linear_Expression.defs.hh.

11.132.4.34 friend class Parma_Polyhedra_Library::Grid_Generator_System [friend]

Definition at line 405 of file Linear_Expression.defs.hh.

11.132.4.35 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Linear_Expression & *e*) [friend]

11.132.4.36 friend class Parma_Polyhedra_Library::Polyhedron [friend]

Definition at line 388 of file Linear_Expression.defs.hh.

11.132.4.37 friend class Parma_Polyhedra_Library::Scalar_Products [friend]

Definition at line 381 of file Linear_Expression.defs.hh.

11.132.4.38 PPL::Linear_Expression & sub_mul_assign (Linear_Expression & *e*, Coefficient_traits::const_reference *n*, Variable *v*) [friend]

Returns the linear expression `e - n * v` and assigns it to `e`.

Definition at line 379 of file Linear_Expression.cc.

11.132.4.39 void swap (Parma_Polyhedra_Library::Linear_Expression & *x*, Parma_Polyhedra_Library::Linear_Expression & *y*) [related]

Specializes `std::swap`.

Definition at line 184 of file Linear_Expression.inlines.hh.

References swap().

11.132.5 Member Data Documentation

11.132.5.1 const PPL::Linear_Expression * Parma_Polyhedra_Library::Linear_Expression::zero_p = 0 [static, private]

Holds (between class initialization and finalization) a pointer to the (zero-dimension space) constant 0.

Definition at line 379 of file Linear_Expression.defs.hh.

Referenced by finalize(), initialize(), and zero().

The documentation for this class was generated from the following files:

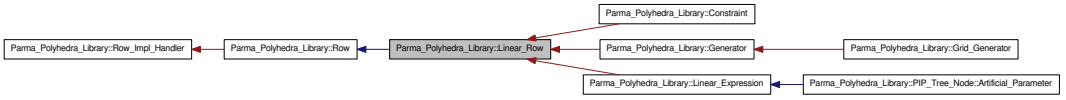
- Linear_Expression.defs.hh
- Linear_Expression.inlines.hh
- Linear_Expression.cc

11.133 Parma_Polyhedra_Library::Linear_Row Class Reference

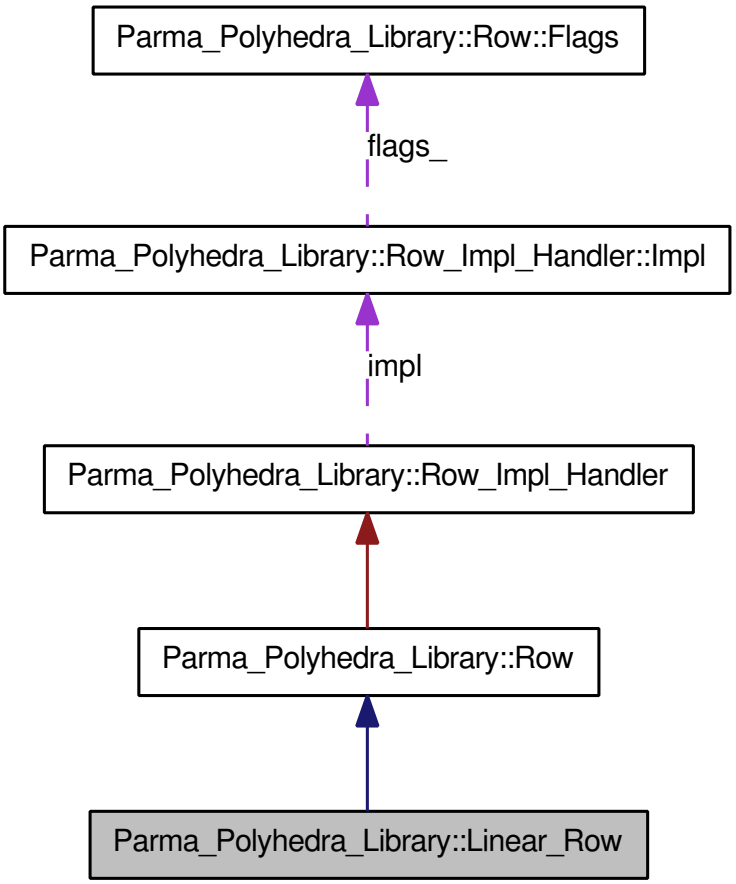
The base class for linear expressions, constraints and generators.

`#include <Linear_Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Linear_Row:



Collaboration diagram for Parma_Polyhedra_Library::Linear_Row:



Classes

- class Flags

  The type of the object to which the coefficients refer to, encoding both topology and kind.

Public Types

- enum Kind { LINE_OR_EQUALITY = 0, RAY_OR_POINT_OR_INEQUALITY = 1 }

  The possible kinds of Linear_Row objects.

Public Member Functions

- Linear_Row ()

  Pre-constructs a row: construction must be completed by construct().

- Linear_Row (dimension_type sz, Flags f)

  Tight constructor: resizing will require reallocation.

- Linear_Row (dimension_type sz, dimension_type capacity, Flags f)

  Sizing constructor with capacity.

- Linear_Row (const Linear_Row &y)

  Ordinary copy constructor.

- Linear_Row (const Linear_Row &y, dimension_type capacity)

  Copy constructor with specified capacity.

- Linear_Row (const Linear_Row &y, dimension_type sz, dimension_type capacity)

  Copy constructor with specified size and capacity.

- ~Linear_Row ()

  Destructor.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Coefficient_traits::const_reference inhomogeneous_term () const

  Returns the inhomogeneous term.

- Coefficient_traits::const_reference coefficient (dimension_type n) const

  Returns the coefficient a_n.

- void sign_normalize ()

  Normalizes the sign of the coefficients so that the first non-zero (homogeneous) coefficient of a line-or-equality is positive.

- void strong_normalize ()

  Strong normalization: ensures that different Linear_Row objects represent different hyperplanes or hyperspaces.

- bool check_strong_normalized () const

  Returns `true` *if and only if the coefficients are strongly normalized.*

- void linear_combine (const Linear_Row &y, dimension_type k)

  Linearly combines `*this` *with* `y` *so that* `*this[k]` *is 0.*

- bool is_zero () const

  Returns `true` *if and only if all the terms of* `*this` *are* 0.

- bool all_homogeneous_terms_are_zero () const

  Returns `true` *if and only if all the homogeneous terms of* `*this` *are* 0.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- bool OK () const

  Checks if all the invariants are satisfied.

- bool OK (dimension_type row_size, dimension_type row_capacity) const

  Checks if all the invariants are satisfied and that the actual size and capacity match the values provided as arguments.

Post-constructors

- void construct (dimension_type sz, Flags f)

  Constructs properly a default-constructed element.

- void construct (dimension_type sz, dimension_type capacity, Flags f)

  Constructs properly a default-constructed element.

Flags inspection methods

- const Flags & flags () const

  Returns a const reference to the flags of `*this`.

- Flags & flags ()

  Returns a non-const reference to the flags of `*this`.

- Topology topology () const

  Returns the topological kind of `*this`.

- bool is_not_necessarily_closed () const

  Returns `true` *if and only if the topology of* `*this` *row is not necessarily closed.*

- bool is_necessarily_closed () const

  Returns `true` *if and only if the topology of* `*this` *row is necessarily closed.*

- bool is_line_or_equality () const

  Returns `true` *if and only if* `*this` *row represents a line or an equality.*

- bool is_ray_or_point_or_inequality () const

  Returns `true` *if and only if* `*this` *row represents a ray, a point or an inequality.*

Flags coercion methods

- void set_necessarily_closed ()

  Sets to `NECESSARILY_CLOSED` *the topological kind of* `*this` *row.*

- void set_not_necessarily_closed ()
  Sets to `NOT_NECESSARILY_CLOSED` *the topological kind of* `*this` *row.*
- void set_is_line_or_equality ()
  Sets to `LINE_OR_EQUALITY` *the kind of* `*this` *row.*
- void set_is_ray_or_point_or_inequality ()
  Sets to `RAY_OR_POINT_OR_INEQUALITY` *the kind of* `*this` *row.*

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension a Linear_Row can handle.

Friends

- class Parma_Polyhedra_Library::Linear_Expression
- class Parma_Polyhedra_Library::Constraint
- class Parma_Polyhedra_Library::Generator

Related Functions

(Note that these are not member functions.)

- bool operator== (const Linear_Row &x, const Linear_Row &y)
  Returns `true` *if and only if* `x` *and* `y` *are equal.*
- bool operator!= (const Linear_Row &x, const Linear_Row &y)
  Returns `true` *if and only if* `x` *and* `y` *are different.*
- int compare (const Linear_Row &x, const Linear_Row &y)
  The basic comparison function.
- void swap (Parma_Polyhedra_Library::Linear_Row &x, Parma_Polyhedra_Library::Linear_Row &y)
  Specializes `std::swap`.
- void iter_swap (std::vector< Parma_Polyhedra_Library::Linear_Row >::iterator x, std::vector< Parma_Polyhedra_Library::Linear_Row >::iterator y)
  Specializes `std::iter_swap`.

11.133.1 Detailed Description

The base class for linear expressions, constraints and generators. The class Linear_Row allows us to build objects of the form $[b, a_0, \ldots, a_{d-1}]_{(t,k)}$, i.e., a finite sequence of coefficients subscripted by a pair of flags, which are both stored in a Linear_Row::Flags object. The flag $t \in \{\mathrm{c}, \mathrm{nnc}\}$ represents the *topology* and the flag $k \in \{=, \geq\}$ represents the *kind* of the Linear_Row object. Note that, even though all the four possible

combinations of topology and kind values will result in a legal Linear_Row::Flags object, some of these pose additional constraints on the values of the Linear_Row's coefficients.

When $t = c$, we have the following cases (d is the dimension of the vector space):

- $[b, a_0, \ldots, a_{d-1}]_{(c,=)}$ represents the equality constraint $\sum_{i=0}^{d-1} a_i x_i + b = 0$.
- $[b, a_0, \ldots, a_{d-1}]_{(c,\geq)}$ represents the non-strict inequality constraint $\sum_{i=0}^{d-1} a_i x_i + b \geq 0$.
- $[0, a_0, \ldots, a_{d-1}]_{(c,=)}$ represents the line of direction $\boldsymbol{l} = (a_0, \ldots, a_{d-1})^{\mathrm{T}}$.
- $[0, a_0, \ldots, a_{d-1}]_{(c,\geq)}$ represents the ray of direction $\boldsymbol{r} = (a_0, \ldots, a_{d-1})^{\mathrm{T}}$.
- $[b, a_0, \ldots, a_{d-1}]_{(c,\geq)}$, with $b > 0$, represents the point $\boldsymbol{p} = (\frac{a_0}{b}, \ldots, \frac{a_{d-1}}{b})^{\mathrm{T}}$.

When $t = \mathrm{nnc}$, the last coefficient of the Linear_Row is associated to the slack variable ϵ, so that we have the following cases (d is again the dimension of the vector space, but this time we have $d + 2$ coefficients):

- $[b, a_0, \ldots, a_{d-1}, 0]_{(\mathrm{nnc},=)}$ represents the equality constraint $\sum_{i=0}^{d-1} a_i x_i + b = 0$.
- $[b, a_0, \ldots, a_{d-1}, 0]_{(\mathrm{nnc},\geq)}$ represents the non-strict inequality constraint $\sum_{i=0}^{d-1} a_i x_i + b \geq 0$.
- $[b, a_0, \ldots, a_{d-1}, e]_{(\mathrm{nnc},\geq)}$, with $e < 0$, represents the strict inequality constraint $\sum_{i=0}^{d-1} a_i x_i + b > 0$.
- $[0, a_0, \ldots, a_{d-1}, 0]_{(\mathrm{nnc},=)}$ represents the line of direction $\boldsymbol{l} = (a_0, \ldots, a_{d-1})^{\mathrm{T}}$.
- $[0, a_0, \ldots, a_{d-1}, 0]_{(\mathrm{nnc},\geq)}$ represents the ray of direction $\boldsymbol{r} = (a_0, \ldots, a_{d-1})^{\mathrm{T}}$.
- $[b, a_0, \ldots, a_{d-1}, e]_{(\mathrm{nnc},\geq)}$, with $b > 0$ and $e > 0$, represents the point $\boldsymbol{p} = (\frac{a_0}{b}, \ldots, \frac{a_{d-1}}{b})^{\mathrm{T}}$.
- $[b, a_0, \ldots, a_{d-1}, 0]_{(\mathrm{nnc},\geq)}$, with $b > 0$, represents the closure point $\boldsymbol{c} = (\frac{a_0}{b}, \ldots, \frac{a_{d-1}}{b})^{\mathrm{T}}$.

So, a Linear_Row can be both a constraint and a generator: it can be an equality, a strict or non-strict inequality, a line, a ray, a point or a closure point.

The inhomogeneous term of a constraint can be zero or different from zero.

Points and closure points must have a positive inhomogeneous term (which is used as a common divisor for all the other coefficients), lines and rays must have the inhomogeneous term equal to zero. If needed, the coefficients of points and closure points are negated at creation time so that they satisfy this invariant. The invariant is maintained because, when combining a point or closure point with another generator, we only consider positive combinations.

The ϵ coefficient, when present, is negative for strict inequality constraints, positive for points and equal to zero in all the other cases. Note that the above description corresponds to the end-user, high-level view of a Linear_Row object. In the implementation, to allow for code reuse, it is sometimes useful to regard an nnc-object on the vector space $\mathbb{R}^d$ as if it was a c-object on the vector space $\mathbb{R}^{d+1}$, therefore interpreting the slack variable ϵ as an ordinary dimension of the vector space.

A Linear_Row object implementing a Linear_Expression is always of the form $[0, a_0, \ldots, a_{d-1}]_{(c,=)}$, which represents the linear expression $\sum_{i=0}^{d-1} a_i x_i$.

Definition at line 125 of file Linear_Row.defs.hh.

11.133.2 Member Enumeration Documentation

11.133.2.1 enum Parma_Polyhedra_Library::Linear_Row::Kind

The possible kinds of Linear_Row objects.

Enumerator:

LINE_OR_EQUALITY
RAY_OR_POINT_OR_INEQUALITY

Definition at line 128 of file Linear_Row.defs.hh.

11.133.3 Constructor & Destructor Documentation

11.133.3.1 Parma_Polyhedra_Library::Linear_Row::Linear_Row () [inline]

Pre-constructs a row: construction must be completed by construct().

Definition at line 159 of file Linear_Row.inlines.hh.

11.133.3.2 Parma_Polyhedra_Library::Linear_Row::Linear_Row (dimension_type *sz*, Flags *f*) [inline]

Tight constructor: resizing will require reallocation.

Definition at line 181 of file Linear_Row.inlines.hh.

References construct().

11.133.3.3 Parma_Polyhedra_Library::Linear_Row::Linear_Row (dimension_type *sz*, dimension_type *capacity*, Flags *f*) [inline]

Sizing constructor with capacity.

Definition at line 170 of file Linear_Row.inlines.hh.

References construct().

11.133.3.4 Parma_Polyhedra_Library::Linear_Row::Linear_Row (const Linear_Row & *y*) [inline]

Ordinary copy constructor.

Definition at line 186 of file Linear_Row.inlines.hh.

11.133.3.5 Parma_Polyhedra_Library::Linear_Row::Linear_Row (const Linear_Row & *y*, dimension_type *capacity*) [inline]

Copy constructor with specified capacity.

It is assumed that `capacity` is greater than or equal to `y` size.

Definition at line 191 of file Linear_Row.inlines.hh.

11.133.3.6 Parma_Polyhedra_Library::Linear_Row::Linear_Row (const Linear_Row & *y*, dimension_type *sz*, dimension_type *capacity*) [inline]

Copy constructor with specified size and capacity.

It is assumed that `sz` is greater than or equal to the size of `y` and, of course, that `sz` is less than or equal to `capacity`.

Definition at line 197 of file Linear_Row.inlines.hh.

11.133.3.7 Parma_Polyhedra_Library::Linear_Row::~Linear_Row () [inline]

Destructor.

Definition at line 203 of file Linear_Row.inlines.hh.

11.133.4 Member Function Documentation

11.133.4.1 bool Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero () const

Returns `true` if and only if all the homogeneous terms of `*this` are 0.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, and Parma_Polyhedra_Library::Grid_Generator.

Definition at line 135 of file Linear_Row.cc.

References Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::Generator::OK(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.133.4.2 void Parma_Polyhedra_Library::Linear_Row::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 190 of file Linear_Row.cc.

References Parma_Polyhedra_Library::Linear_Row::Flags::ascii_dump(), flags(), and Parma_Polyhedra_Library::Row::size().

11.133.4.3 void Parma_Polyhedra_Library::Linear_Row::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

11.133.4.4 bool Parma_Polyhedra_Library::Linear_Row::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 204 of file Linear_Row.cc.

References Parma_Polyhedra_Library::Row::shrink(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Row::swap().

11.133.4.5 bool Parma_Polyhedra_Library::Linear_Row::check_strong_normalized () const

Returns `true` if and only if the coefficients are strongly normalized.

Definition at line 58 of file Linear_Row.cc.

References compare(), and strong_normalize().

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Linear_System::add_row(), Parma_Polyhedra_Library::Linear_System::insert(), and Parma_Polyhedra_Library::Linear_System::insert_pending().

11.133.4.6 Coefficient_traits::const_reference Parma_Polyhedra_Library::Linear_Row::coefficient (dimension_type *n*) const [inline]

Returns the coefficient a_n.

Definition at line 247 of file Linear_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Polyhedron::constrains().

11.133.4.7 void Parma_Polyhedra_Library::Linear_Row::construct (dimension_type *sz*, dimension_type *capacity*, Flags *f*) [inline]

Constructs properly a default-constructed element.

Parameters

sz The size of the row that will be constructed;

capacity The minimum capacity of the row that will be constructed.

f Flags for the row that will be constructed.

The row that we are constructing has a minimum capacity, i.e., it can contain at least `capacity` elements, `sz` of which will be default-constructed now. The row flags are set to `f`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 164 of file Linear_Row.inlines.hh.

References construct().

11.133.4.8 void Parma_Polyhedra_Library::Linear_Row::construct (dimension_type *sz*, Flags *f*) [inline]

Constructs properly a default-constructed element.

Builds a row with type `t`, size `sz` and minimum capacity.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 176 of file Linear_Row.inlines.hh.

Referenced by construct(), Parma_Polyhedra_Library::Linear_Expression::Linear_Expression(), and Linear_Row().

11.133.4.9 Linear_Row::Flags & Parma_Polyhedra_Library::Linear_Row::flags () [inline]

Returns a non-const reference to the flags of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 134 of file Linear_Row.inlines.hh.

References flags().

11.133.4.10 const Linear_Row::Flags & Parma_Polyhedra_Library::Linear_Row::flags () const [inline]

Returns a const reference to the flags of `*this`.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 129 of file Linear_Row.inlines.hh.

Referenced by ascii_dump(), Parma_Polyhedra_Library::Constraint::Constraint(), flags(), Parma_Polyhedra_Library::Generator::Generator(), is_line_or_equality(), is_necessarily_closed(), is_ray_or_point_or_inequality(), operator==(), set_is_line_or_equality(), set_is_ray_or_point_or_inequality(), set_necessarily_closed(), set_not_necessarily_closed(), and topology().

11.133.4.11 Coefficient_traits::const_reference Parma_Polyhedra_Library::Linear_Row::inhomogeneous_term () const [inline]

Returns the inhomogeneous term.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, and Parma_Polyhedra_Library::Constraint.

Definition at line 242 of file Linear_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Generator::divisor().

11.133.4.12 bool Parma_Polyhedra_Library::Linear_Row::is_line_or_equality () const [inline]

Returns `true` if and only if `*this` row represents a line or an equality.

Definition at line 207 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), compare(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Generator::is_line(), and sign_normalize().

11.133.4.13 bool Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed () const [inline]

Returns `true` if and only if the topology of `*this` row is necessarily closed.

Definition at line 139 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::is_necessarily_closed().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Generator::is_equivalent_to(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::Grid_Generator::OK(), Parma_Polyhedra_Library::Generator::OK(), Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator()(), Parma_Polyhedra_Library::Scalar_Products::reduced_assign(), Parma_Polyhedra_Library::Polyhedron::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), space_dimension(), Parma_Polyhedra_Library::Generator::type(), and Parma_Polyhedra_Library::Constraint::type().

11.133.4.14 bool Parma_Polyhedra_Library::Linear_Row::is_not_necessarily_closed () const

Returns `true` if and only if the topology of `*this` row is not necessarily closed.

11.133.4.15 bool Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality () const [inline]

Returns `true` if and only if `*this` row represents a ray, a point or an inequality.

Definition at line 212 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::is_ray_or_point_or_inequality().

Referenced by Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Grid_Generator::is_parameter_or_point(), and Parma_Polyhedra_Library::Generator::is_ray_or_point().

11.133.4.16 bool Parma_Polyhedra_Library::Linear_Row::is_zero () const

Returns `true` if and only if all the terms of `*this` are 0.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression.

Definition at line 126 of file Linear_Row.cc.

References Parma_Polyhedra_Library::Row::size().

11.133.4.17 void Parma_Polyhedra_Library::Linear_Row::linear_combine (const Linear_Row & *y*, dimension_type *k*)

Linearly combines `*this` with `y` so that `*this[k]` is 0.

Parameters

- ***y*** The Linear_Row that will be combined with `*this` object;
- ***k*** The position of `*this` that have to be 0.

Computes a linear combination of `*this` and `y` having the element of index `k` equal to 0. Then it assigns the resulting Linear_Row to `*this` and normalizes it.

Definition at line 104 of file Linear_Row.cc.

References Parma_Polyhedra_Library::normalize2(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Row::size(), strong_normalize(), and Parma_Polyhedra_Library::sub_mul_assign().

Referenced by Parma_Polyhedra_Library::Linear_System::back_substitute(), and Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points().

11.133.4.18 dimension_type Parma_Polyhedra_Library::Linear_Row::max_space_dimension () [inline, static]

Returns the maximum space dimension a Linear_Row can handle.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, and Parma_Polyhedra_Library::Grid_Generator.

Definition at line 144 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::Row::max_size().

11.133.4.19 bool Parma_Polyhedra_Library::Linear_Row::OK (dimension_type *row_size*, dimension_type *row_capacity*) const

Checks if all the invariants are satisfied and that the actual size and capacity match the values provided as arguments.

Reimplemented from Parma_Polyhedra_Library::Row.

Definition at line 235 of file Linear_Row.cc.

References OK().

11.133.4.20 bool Parma_Polyhedra_Library::Linear_Row::OK () const

Checks if all the invariants are satisfied.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 230 of file Linear_Row.cc.

Referenced by OK(), Parma_Polyhedra_Library::Generator::OK(), and Parma_Polyhedra_Library::Constraint::OK().

11.133.4.21 void Parma_Polyhedra_Library::Linear_Row::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Row.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

11.133.4.22 void Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality () [inline]

Sets to `LINE_OR_EQUALITY` the kind of `*this` row.

Definition at line 222 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::set_is_line_or_equality().

Referenced by Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Constraint::set_is_equality(), Parma_-

Polyhedra_Library::Generator::set_is_line(), and Parma_Polyhedra_Library::Polyhedron::simplify_-using_context_assign().

11.133.4.23 void Parma_Polyhedra_Library::Linear_Row::set_is_ray_or_point_or_inequality () [inline]

Sets to RAY_OR_POINT_OR_INEQUALITY the kind of *this row.

Definition at line 227 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::set_is_ray_or_point_or_-inequality().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_-Polyhedra_Library::Constraint::set_is_inequality(), and Parma_Polyhedra_Library::Generator::set_is_-ray_or_point().

11.133.4.24 void Parma_Polyhedra_Library::Linear_Row::set_necessarily_closed () [inline]

Sets to NECESSARILY_CLOSED the topological kind of *this row.

Definition at line 232 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::set_necessarily_closed().

11.133.4.25 void Parma_Polyhedra_Library::Linear_Row::set_not_necessarily_closed () [inline]

Sets to NOT_NECESSARILY_CLOSED the topological kind of *this row.

Definition at line 237 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::set_not_necessarily_closed().

11.133.4.26 void Parma_Polyhedra_Library::Linear_Row::sign_normalize ()

Normalizes the sign of the coefficients so that the first non-zero (homogeneous) coefficient of a line-or-equality is positive.

Definition at line 34 of file Linear_Row.cc.

References is_line_or_equality(), Parma_Polyhedra_Library::neg_assign(), and Parma_Polyhedra_-Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_-Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and strong_normalize().

11.133.4.27 dimension_type Parma_Polyhedra_Library::Linear_Row::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Generator, and Parma_Polyhedra_Library::Grid_Generator.

Definition at line 151 of file Linear_Row.inlines.hh.

References is_necessarily_closed(), and Parma_Polyhedra_Library::Row::size().

11.133.4.28 void Parma_Polyhedra_Library::Linear_Row::strong_normalize () [inline]

Strong normalization: ensures that different Linear_Row objects represent different hyperplanes or hyperspaces.

Applies both Linear_Row::normalize() and Linear_Row::sign_normalize().

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 252 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::Row::normalize(), and sign_normalize().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_congruence(), Parma_Polyhedra_Library::Polyhedron::add_congruences(), check_strong_normalized(), Parma_Polyhedra_Library::Constraint::Constraint(), Parma_Polyhedra_Library::Polyhedron::conversion(), linear_combine(), Parma_Polyhedra_Library::Generator::OK(), Parma_Polyhedra_Library::Constraint::OK(), Parma_Polyhedra_Library::Polyhedron::refine_with_congruence(), and Parma_Polyhedra_Library::Polyhedron::refine_with_congruences().

11.133.4.29 Topology Parma_Polyhedra_Library::Linear_Row::topology () const [inline]

Returns the topological kind of `*this`.

Definition at line 217 of file Linear_Row.inlines.hh.

References flags(), and Parma_Polyhedra_Library::Linear_Row::Flags::topology().

Referenced by Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), and Parma_Polyhedra_Library::Generator::is_matching_closure_point().

11.133.5 Friends And Related Function Documentation

11.133.5.1 int compare (const Linear_Row & *x*, const Linear_Row & *y*) [related]

The basic comparison function.

Returns

The returned absolute value can be 0, 1 or 2.

Parameters

x A row of coefficients;

y Another row.

Compares `x` and `y`, where `x` and `y` may be of different size, in which case the "missing" coefficients are assumed to be zero. The comparison is such that:

1. equalities are smaller than inequalities;
2. lines are smaller than points and rays;
3. the ordering is lexicographic;
4. the positions compared are, in decreasing order of significance, 1, 2, ..., `size()`, 0;
5. the result is negative, zero, or positive if x is smaller than, equal to, or greater than y, respectively;
6. when `x` and `y` are different, the absolute value of the result is 1 if the difference is due to the coefficient in position 0; it is 2 otherwise.

When `x` and `y` represent the hyper-planes associated to two equality or inequality constraints, the coefficient at 0 is the known term. In this case, the return value can be characterized as follows:

- -2, if `x` is smaller than `y` and they are *not* parallel;
- -1, if `x` is smaller than `y` and they *are* parallel;
- 0, if `x` and y are equal;
- +1, if `y` is smaller than `x` and they *are* parallel;
- +2, if `y` is smaller than `x` and they are *not* parallel.

Definition at line 66 of file Linear_Row.cc.

References Parma_Polyhedra_Library::cmp(), is_line_or_equality(), and Parma_Polyhedra_Library::Row::size().

Referenced by check_strong_normalized().

11.133.5.2 void iter_swap (std::vector< Parma_Polyhedra_Library::Linear_Row >::iterator *x*, std::vector< Parma_Polyhedra_Library::Linear_Row >::iterator *y*) [related]

Specializes `std::iter_swap`.

Definition at line 284 of file Linear_Row.inlines.hh.

References Parma_Polyhedra_Library::swap().

11.133.5.3 bool operator!= (const Linear_Row & *x*, const Linear_Row & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 266 of file Linear_Row.inlines.hh.

11.133.5.4 bool operator== (const Linear_Row & *x*, const Linear_Row & *y*) [related]

Returns `true` if and only if `x` and `y` are equal.

Definition at line 259 of file Linear_Row.inlines.hh.

References flags().

11.133.5.5 friend class Parma_Polyhedra_Library::Constraint [friend]

Reimplemented in Parma_Polyhedra_Library::Linear_Expression.

Definition at line 395 of file Linear_Row.defs.hh.

11.133.5.6 friend class Parma_Polyhedra_Library::Generator [friend]

Reimplemented in Parma_Polyhedra_Library::Linear_Expression.

Definition at line 396 of file Linear_Row.defs.hh.

11.133.5.7 friend class Parma_Polyhedra_Library::Linear_Expression [friend]

Definition at line 394 of file Linear_Row.defs.hh.

11.133.5.8 void swap (Parma_Polyhedra_Library::Linear_Row & *x*, Parma_Polyhedra_Library::Linear_Row & *y*) [related]

Specializes `std::swap`.

Definition at line 277 of file Linear_Row.inlines.hh.

References swap().

Referenced by swap().

The documentation for this class was generated from the following files:

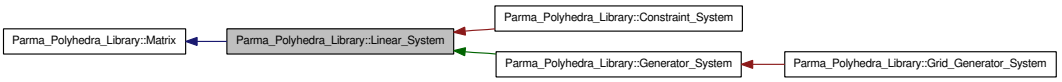
- Linear_Row.defs.hh
- Linear_Row.inlines.hh
- Linear_Row.cc

11.134 Parma_Polyhedra_Library::Linear_System Class Reference

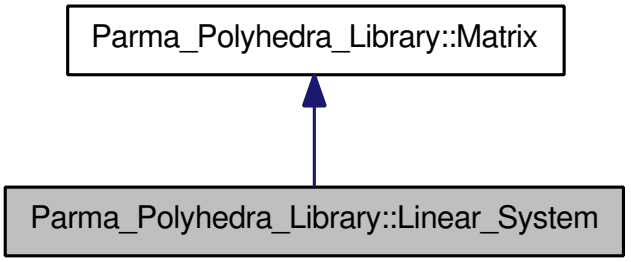
The base class for systems of constraints and generators.

`#include <Linear_System.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Linear_System:



Collaboration diagram for Parma_Polyhedra_Library::Linear_System:



Classes

- struct Row_Less_Than

  Ordering predicate (used when implementing the sort algorithm).

- struct With_Pending

  A tag class.

Public Member Functions

- Linear_System (Topology topol)

  Builds an empty linear system with specified topology.

- Linear_System (Topology topol, dimension_type n_rows, dimension_type n_columns)

  Builds a system with specified topology and dimensions.

- Linear_System (const Linear_System &y)

  Copy constructor: pending rows are transformed into non-pending ones.

- Linear_System (const Linear_System &y, With_Pending)

  Full copy constructor: pending rows are copied as pending.

- Linear_System & operator= (const Linear_System &y)

  Assignment operator: pending rows are transformed into non-pending ones.

- void assign_with_pending (const Linear_System &y)

  Full assignment operator: pending rows are copied as pending.

- void swap (Linear_System &y)

  Swaps `*this` *with* `y`.

- dimension_type space_dimension () const

  Returns the space dimension of the rows in the system.

- void remove_trailing_columns (dimension_type n)

  Makes the system shrink by removing its `n` *trailing columns.*

- void permute_columns (const std::vector< dimension_type > &cycles)

  Permutes the columns of the system.

- void strong_normalize ()

  Strongly normalizes the system.

- void sign_normalize ()

  Sign-normalizes the system.

- bool check_sorted () const

  Returns `true` *if and only if* `*this` *is sorted, without checking for duplicates.*

- void set_necessarily_closed ()

  Sets the system topology to `NECESSARILY_CLOSED`.

- void set_not_necessarily_closed ()

  Sets the system topology to `NOT_NECESSARILY_CLOSED`.

- void set_rows_topology ()

  Sets the topology of all rows equal to the system topology.

- void unset_pending_rows ()

  Sets the index to indicate that the system has no pending rows.

- void set_index_first_pending_row (dimension_type i)

  Sets the index of the first pending row to `i`.

- void set_sorted (bool b)

  Sets the sortedness flag of the system to `b`.

- void resize_no_copy (dimension_type new_n_rows, dimension_type new_n_columns)

  Resizes the system without worrying about the old contents.

- void add_rows_and_columns (dimension_type n)

  Adds `n` *rows and columns to the system.*

- void insert (const Linear_Row &r)

  Adds a copy of `r` *to the system, automatically resizing the system or the row's copy, if needed.*

- void insert_pending (const Linear_Row &r)

  Adds a copy of the given row to the pending part of the system, automatically resizing the system or the row, if needed.

- void add_row (const Linear_Row &r)

  Adds a copy of the given row to the system.

- void add_pending_row (Linear_Row::Flags flags)

  Adds a new empty row to the system, setting only its flags.

- void add_pending_row (const Linear_Row &r)

  Adds a copy of the given row to the pending part of the system.

- void add_rows (const Linear_System &y)

  Adds to `*this` a copy of the rows of 'y'.

- void add_pending_rows (const Linear_System &y)

  *Adds a copy of the rows of 'y' to the pending part of '*this'.*

- void sort_rows ()

  Sorts the non-pending rows (in growing order) and eliminates duplicated ones.

- void sort_rows (dimension_type first_row, dimension_type last_row)

  Sorts the rows (in growing order) form `first_row` to `last_row` and eliminates duplicated ones.

- void merge_rows_assign (const Linear_System &y)

  Assigns to `*this` the result of merging its rows with those of `y`, obtaining a sorted system.

- void sort_pending_and_remove_duplicates ()

  Sorts the pending rows and eliminates those that also occur in the non-pending part of the system.

- void sort_and_remove_with_sat (Bit_Matrix &sat)

  Sorts the system, removing duplicates, keeping the saturation matrix consistent.

- dimension_type gauss (dimension_type n_lines_or_equalities)

  Minimizes the subsystem of equations contained in `*this`.

- void back_substitute (dimension_type n_lines_or_equalities)

  Back-substitutes the coefficients to reduce the complexity of the system.

- void simplify ()

  Applies Gaussian elimination and back-substitution so as to simplify the linear system.

- void normalize ()

  Normalizes the system by dividing each row for the GCD of the row's elements.

- void clear ()

  Clears the system deallocating all its rows.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK (bool check_strong_normalized=true) const

  Checks if all the invariants are satisfied.

Subscript operators

- Linear_Row & operator[ ] (dimension_type k)

  Returns a reference to the `k-th` row of the system.

- const Linear_Row & operator[ ] (dimension_type k) const

  Returns a constant reference to the `k-th` row of the system.

Accessors

- Topology topology () const

  Returns the system topology.

- bool is_sorted () const

  Returns the value of the sortedness flag.

- bool is_necessarily_closed () const

  Returns `true` if and only if the system topology is `NECESSARILY_CLOSED`.

- dimension_type num_lines_or_equalities () const

  Returns the number of rows in the system that represent either lines or equalities.

- dimension_type first_pending_row () const

  Returns the index of the first pending row.

- dimension_type num_pending_rows () const

  Returns the number of rows that are in the pending part of the system.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Linear_System can handle.

Private Attributes

- Topology row_topology

  The topological kind of the rows in the system.

- dimension_type index_first_pending

  The index of the first pending row.

- bool sorted

  `true` *if rows are sorted in the ascending order as defined by* `bool compare(const Linear_Row&, const Linear_Row&)`. *If* `false` *may not be sorted.*

Related Functions

(Note that these are not member functions.)

- void swap (Parma_Polyhedra_Library::Linear_System &x, Parma_Polyhedra_Library::Linear_System &y)

  Specializes `std::swap`.

- bool operator== (const Linear_System &x, const Linear_System &y)

  Returns `true` *if and only if* `x` *and* `y` *are identical.*

- bool operator!= (const Linear_System &x, const Linear_System &y)

  Returns `true` *if and only if* `x` *and* `y` *are different.*

11.134.1 Detailed Description

The base class for systems of constraints and generators. An object of this class represents either a constraint system or a generator system. Each Linear_System object can be viewed as a finite sequence of strong-normalized Linear_Row objects, where each Linear_Row implements a constraint or a generator. Linear systems are characterized by the matrix of coefficients, also encoding the number, size and capacity of Linear_row objects, as well as a few additional information, including:

- the topological kind of (all) the rows;
- an indication of whether or not some of the rows in the Linear_System are *pending*, meaning that they still have to undergo an (unspecified) elaboration; if there are pending rows, then these form a proper suffix of the overall sequence of rows;
- a Boolean flag that, when `true`, ensures that the non-pending prefix of the sequence of rows is sorted.

Definition at line 55 of file Linear_System.defs.hh.

11.134.2 Constructor & Destructor Documentation

11.134.2.1 Parma_Polyhedra_Library::Linear_System::Linear_System (Topology *topol*) [inline]

Builds an empty linear system with specified topology.

Rows size and capacity are initialized to 0.

Definition at line 57 of file Linear_System.inlines.hh.

11.134.2.2 Parma_Polyhedra_Library::Linear_System::Linear_System (Topology *topol*, dimension_type *n_rows*, dimension_type *n_columns*) [inline]

Builds a system with specified topology and dimensions.

Parameters

topol The topology of the system that will be created;

n_rows The number of rows of the system that will be created;

n_columns The number of columns of the system that will be created.

Creates a `n_rows` × `n_columns` system whose coefficients are all zero and whose rows are all initialized to be of the given topology.

Definition at line 65 of file Linear_System.inlines.hh.

11.134.2.3 Parma_Polyhedra_Library::Linear_System::Linear_System (const Linear_System & *y*) [inline]

Copy constructor: pending rows are transformed into non-pending ones.

Definition at line 95 of file Linear_System.inlines.hh.

References num_pending_rows(), sorted, and unset_pending_rows().

11.134.2.4 Parma_Polyhedra_Library::Linear_System::Linear_System (const Linear_System & *y*, With_Pending) [inline]

Full copy constructor: pending rows are copied as pending.

Definition at line 105 of file Linear_System.inlines.hh.

11.134.3 Member Function Documentation

11.134.3.1 void Parma_Polyhedra_Library::Linear_System::add_pending_row (const Linear_Row & *r*)

Adds a copy of the given row to the pending part of the system.

Definition at line 397 of file Linear_System.cc.

References Parma_Polyhedra_Library::Linear_Row::check_strong_normalized(), Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::Matrix::max_num_rows(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, Parma_Polyhedra_Library::Matrix::rows, Parma_Polyhedra_Library::Row::size(), and swap().

11.134.3.2 void Parma_Polyhedra_Library::Linear_System::add_pending_row (Linear_Row::Flags *flags*)

Adds a new empty row to the system, setting only its flags.

Definition at line 435 of file Linear_System.cc.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::construct(), Parma_Polyhedra_Library::Matrix::max_num_rows(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, Parma_Polyhedra_Library::Matrix::rows, and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Generator_System::add_corresponding_points(), Parma_Polyhedra_Library::Polyhedron::conversion(), and insert_pending().

11.134.3.3 void Parma_Polyhedra_Library::Linear_System::add_pending_rows (const Linear_System & *y*)

Adds a copy of the rows of 'y' to the pending part of '∗this'.

Definition at line 265 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, row_topology, sorted, and swap().

Referenced by add_rows(), Parma_Polyhedra_Library::Polyhedron::intersection_assign(), Parma_Polyhedra_Library::Polyhedron::poly_hull_assign(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.134.3.4 void Parma_Polyhedra_Library::Linear_System::add_row (const Linear_Row & *r*)

Adds a copy of the given row to the system.

Definition at line 359 of file Linear_System.cc.

References Parma_Polyhedra_Library::Linear_Row::check_strong_normalized(), is_sorted(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_size, set_index_first_pending_row(),

set_sorted(), and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), insert(), and Parma_Polyhedra_Library::Grid_Generator_System::insert().

11.134.3.5 void Parma_Polyhedra_Library::Linear_System::add_rows (const Linear_System & *y*)

Adds to `*this` a copy of the rows of 'y'.

It is assumed that `*this` has no pending rows.

Definition at line 287 of file Linear_System.cc.

References add_pending_rows(), Parma_Polyhedra_Library::Matrix::has_no_rows(), is_sorted(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), set_sorted(), and unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::intersection_assign(), and Parma_Polyhedra_Library::Polyhedron::poly_hull_assign().

11.134.3.6 void Parma_Polyhedra_Library::Linear_System::add_rows_and_columns (dimension_type *n*)

Adds `n` rows and columns to the system.

Parameters

n The number of rows and columns to be added: must be strictly positive.

Turns the system $M \in \mathbb{R}^r \times \mathbb{R}^c$ into the system $N \in \mathbb{R}^{r+n} \times \mathbb{R}^{c+n}$ such that $N = \begin{pmatrix} 0 & J \\ M & o \end{pmatrix}$, where J is the specular image of the $n \times n$ identity matrix.

Definition at line 748 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns(), is_sorted(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), row_topology, Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality(), set_sorted(), and Parma_Polyhedra_Library::swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), and Parma_Polyhedra_Library::Polyhedron::concatenate_assign().

11.134.3.7 void Parma_Polyhedra_Library::Linear_System::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 114 of file Linear_System.cc.

References ascii_dump(), first_pending_row(), is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), and sorted.

11.134.3.8 void Parma_Polyhedra_Library::Linear_System::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Referenced by ascii_dump().

11.134.3.9 bool Parma_Polyhedra_Library::Linear_System::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reads into a Linear_System object the information produced by the output of ascii_dump(std::ostream&) const. The specialized methods provided by Constraint_System and Generator_System take care of properly reading the contents of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 138 of file Linear_System.cc.

References Parma_Polyhedra_Library::ascii_load().

11.134.3.10 void Parma_Polyhedra_Library::Linear_System::assign_with_pending (const Linear_System & *y*) `[inline]`

Full assignment operator: pending rows are copied as pending.

Definition at line 124 of file Linear_System.inlines.hh.

References index_first_pending, operator=(), row_topology, and sorted.

Referenced by Parma_Polyhedra_Library::Polyhedron::operator=(), and Parma_Polyhedra_Library::Polyhedron::Polyhedron().

11.134.3.11 void Parma_Polyhedra_Library::Linear_System::back_substitute (dimension_type *n_lines_or_equalities*)

Back-substitutes the coefficients to reduce the complexity of the system.

Takes an upper triangular system having `n_lines_or_equalities` rows. For each row, starting from the one having the minimum number of coefficients different from zero, computes the expression of an element as a function of the remaining ones and then substitutes this expression in all the other rows.

Definition at line 607 of file Linear_System.cc.

References is_sorted(), Parma_Polyhedra_Library::Linear_Row::linear_combine(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), num_lines_or_equalities(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), and set_sorted().

Referenced by Parma_Polyhedra_Library::Polyhedron::simplify(), and simplify().

11.134.3.12 bool Parma_Polyhedra_Library::Linear_System::check_sorted () const

Returns `true` if and only if `*this` is sorted, without checking for duplicates.

Definition at line 839 of file Linear_System.cc.

References first_pending_row().

Referenced by is_sorted(), merge_rows_assign(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators_with_sat_g(), OK(), and sort_and_remove_with_sat().

11.134.3.13 void Parma_Polyhedra_Library::Linear_System::clear () [inline]

Clears the system deallocating all its rows.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 140 of file Linear_System.inlines.hh.

References index_first_pending, and sorted.

11.134.3.14 memory_size_type Parma_Polyhedra_Library::Linear_System::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 32 of file Linear_System.inlines.hh.

Referenced by total_memory_in_bytes().

11.134.3.15 dimension_type Parma_Polyhedra_Library::Linear_System::first_pending_row () const [inline]

Returns the index of the first pending row.

Definition at line 74 of file Linear_System.inlines.hh.

References index_first_pending.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), check_sorted(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::is_universe(), num_pending_rows(), Parma_Polyhedra_Library::Polyhedron::OK(), OK(), operator==(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), sort_and_remove_with_sat(), sort_pending_and_remove_duplicates(), sort_rows(), Parma_Polyhedra_Library::Polyhedron::update_sat_c(), and Parma_Polyhedra_Library::Polyhedron::update_sat_g().

11.134.3.16 PPL::dimension_type Parma_Polyhedra_Library::Linear_System::gauss (dimension_type *n_lines_or_equalities*)

Minimizes the subsystem of equations contained in `*this`.

This method works only on the equalities of the system: the system is required to be partially sorted, so that all the equalities are grouped at its top; it is assumed that the number of equalities is exactly `n_lines_or_equalities`. The method finds a minimal system for the equalities and returns its rank, i.e., the number of linearly independent equalities. The result is an upper triangular subsystem of equalities: for each equality, the pivot is chosen starting from the right-most columns.

Definition at line 555 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::num_columns(), num_lines_or_equalities(), num_pending_rows(), OK(), set_sorted(), and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::simplify(), and simplify().

11.134.3.17 void Parma_Polyhedra_Library::Linear_System::insert (const Linear_Row & *r*)

Adds a copy of `r` to the system, automatically resizing the system or the row's copy, if needed.

Definition at line 183 of file Linear_System.cc.

References add_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_Row::check_strong_normalized(), is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Row::size(), swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Linear_Row::topology(), and topology().

Referenced by Parma_Polyhedra_Library::Generator_System::insert(), and Parma_Polyhedra_Library::Constraint_System::insert().

11.134.3.18 void Parma_Polyhedra_Library::Linear_System::insert_pending (const Linear_Row & *r*)

Adds a copy of the given row to the pending part of the system, automatically resizing the system or the row, if needed.

Definition at line 224 of file Linear_System.cc.

References add_pending_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_Row::check_strong_normalized(), is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Row::size(), swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Linear_Row::topology(), and topology().

Referenced by Parma_Polyhedra_Library::Generator_System::insert_pending(), and Parma_Polyhedra_Library::Constraint_System::insert_pending().

11.134.3.19 bool Parma_Polyhedra_Library::Linear_System::is_necessarily_closed () const [inline]

Returns `true` if and only if the system topology is `NECESSARILY_CLOSED`.

Definition at line 174 of file Linear_System.inlines.hh.

References Parma_Polyhedra_Library::NECESSARILY_CLOSED, and row_topology.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Generator_System::add_corresponding_points(), Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator_System::has_closure_points(), Parma_Polyhedra_Library::Generator_System::has_points(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed(), Parma_Polyhedra_Library::Polyhedron::minimize(), OK(), Parma_Polyhedra_Library::Generator_System::const_iterator::operator++(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), set_rows_topology(), and space_dimension().

11.134.3.20 bool Parma_Polyhedra_Library::Linear_System::is_sorted () const [inline]

Returns the value of the sortedness flag.

Definition at line 42 of file Linear_System.inlines.hh.

References check_sorted(), and sorted.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), add_row(),

add_rows(), add_rows_and_columns(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), back_substitute(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::intersection_assign(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Constraint_System::num_inequalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Generator_System::num_rays(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators_with_sat_g(), Parma_Polyhedra_Library::Grid_Generator_System::OK(), Parma_Polyhedra_Library::Polyhedron::poly_hull_assign(), Parma_Polyhedra_Library::Polyhedron::process_pending_constraints(), Parma_Polyhedra_Library::Polyhedron::process_pending_generators(), sort_pending_and_remove_duplicates(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.134.3.21 dimension_type Parma_Polyhedra_Library::Linear_System::max_space_dimension () [inline, static]

Returns the maximum space dimension a Linear_System can handle.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 194 of file Linear_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::max_num_columns().

11.134.3.22 void Parma_Polyhedra_Library::Linear_System::merge_rows_assign (const Linear_System & *y*)

Assigns to `*this` the result of merging its rows with those of `y`, obtaining a sorted system.

Duplicated rows will occur only once in the result. On entry, both systems are assumed to be sorted and have no pending rows.

Definition at line 52 of file Linear_System.cc.

References check_sorted(), Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::Matrix::max_num_rows(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, Parma_Polyhedra_Library::Matrix::rows, swap(), and unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::intersection_assign(), Parma_Polyhedra_Library::Polyhedron::poly_hull_assign(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.134.3.23 void Parma_Polyhedra_Library::Linear_System::normalize ()

Normalizes the system by dividing each row for the GCD of the row's elements.

Definition at line 465 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), and set_sorted().

Referenced by Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension().

11.134.3.24 PPL::dimension_type Parma_Polyhedra_Library::Linear_System::num_lines_or_equalities () const

Returns the number of rows in the system that represent either lines or equalities.

Definition at line 41 of file Linear_System.cc.

References num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), back_substitute(), and gauss().

11.134.3.25 dimension_type Parma_Polyhedra_Library::Linear_System::num_pending_rows () const [inline]

Returns the number of rows that are in the pending part of the system.

Definition at line 79 of file Linear_System.inlines.hh.

References first_pending_row(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), add_pending_row(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), add_row(), add_rows(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), back_substitute(), gauss(), insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), insert_pending(), Linear_System(), merge_rows_assign(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Constraint_System::num_inequalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), num_lines_or_equalities(), Parma_Polyhedra_Library::Generator_System::num_rays(), Parma_Polyhedra_Library::Polyhedron::OK(), operator=(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Polyhedron::process_pending_constraints(), Parma_Polyhedra_Library::Polyhedron::process_pending_generators(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), Parma_Polyhedra_Library::Polyhedron::simplified_constraints(), simplify(), sort_and_remove_with_sat(), sort_pending_and_remove_duplicates(), and sort_rows().

11.134.3.26 bool Parma_Polyhedra_Library::Linear_System::OK (bool *check_strong_normalized* = true) const

Checks if all the invariants are satisfied.

Parameters

check_strong_normalized `true` if and only if the strong normalization of all the rows in the system has to be checked.

By default, the strong normalization check is performed. This check may be turned off to avoid useless repeated checking; e.g., when re-checking a well-formed Linear_System after the permutation or deletion of some of its rows.

Definition at line 848 of file Linear_System.cc.

References check_sorted(), first_pending_row(), Parma_Polyhedra_Library::Matrix::has_no_rows(), is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), Parma_Polyhedra_Library::Matrix::row_capacity, Parma_Polyhedra_Library::Matrix::row_size, sorted, strong_normalize(), and topology().

Referenced by back_substitute(), gauss(), Parma_Polyhedra_Library::Polyhedron::simplify(), and simplify().

11.134.3.27 Linear_System & Parma_Polyhedra_Library::Linear_System::operator= (const Linear_System & *y*) [inline]

Assignment operator: pending rows are transformed into non-pending ones.

Definition at line 113 of file Linear_System.inlines.hh.

References num_pending_rows(), row_topology, sorted, and unset_pending_rows().

Referenced by assign_with_pending().

11.134.3.28 const Linear_Row & Parma_Polyhedra_Library::Linear_System::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the `k-th` row of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 184 of file Linear_System.inlines.hh.

References operator[ ]().

11.134.3.29 Linear_Row & Parma_Polyhedra_Library::Linear_System::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the `k-th` row of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 179 of file Linear_System.inlines.hh.

Referenced by operator[ ]().

11.134.3.30 void Parma_Polyhedra_Library::Linear_System::permute_columns (const std::vector< dimension_type > & *cycles*) [inline]

Permutes the columns of the system.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 218 of file Linear_System.inlines.hh.

References sign_normalize().

Referenced by Parma_Polyhedra_Library::Polyhedron::bounded_affine_preimage(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), and Parma_Polyhedra_Library::Grid_Generator_System::permute_columns().

11.134.3.31 void Parma_Polyhedra_Library::Linear_System::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

11.134.3.32 void Parma_Polyhedra_Library::Linear_System::remove_trailing_columns (dimension_type *n*) [inline]

Makes the system shrink by removing its `n` trailing columns.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Definition at line 210 of file Linear_System.inlines.hh.

References strong_normalize().

Referenced by Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Grid_Generator_System::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions(), and Parma_Polyhedra_Library::Grid_Generator_System::remove_space_dimensions().

11.134.3.33 void Parma_Polyhedra_Library::Linear_System::resize_no_copy (dimension_type *new_n_rows*, dimension_type *new_n_columns*) [inline]

Resizes the system without worrying about the old contents.

Parameters

new_n_rows The number of rows of the resized system;

new_n_columns The number of columns of the resized system.

The system is expanded to the specified dimensions avoiding reallocation whenever possible. The contents of the original system is lost.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 148 of file Linear_System.inlines.hh.

References row_topology, and set_sorted().

Referenced by Parma_Polyhedra_Library::Polyhedron::minimize().

11.134.3.34 void Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row (dimension_type *i*) [inline]

Sets the index of the first pending row to `i`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 90 of file Linear_System.inlines.hh.

References index_first_pending.

Referenced by add_row(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), sort_and_remove_with_sat(), and sort_rows().

11.134.3.35 void Parma_Polyhedra_Library::Linear_System::set_necessarily_closed () [inline]

Sets the system topology to `NECESSARILY_CLOSED`.

Definition at line 160 of file Linear_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, row_topology, and set_rows_topology().

Referenced by Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), set_rows_topology(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.134.3.36 void Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed () [inline]

Sets the system topology to `NOT_NECESSARILY_CLOSED`.

Definition at line 167 of file Linear_System.inlines.hh.

References Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, row_topology, and set_rows_topology().

Referenced by Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), set_rows_topology(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.134.3.37 void Parma_Polyhedra_Library::Linear_System::set_rows_topology ()

Sets the topology of all rows equal to the system topology.

Definition at line 103 of file Linear_System.cc.

References is_necessarily_closed(), Parma_Polyhedra_Library::Matrix::num_rows(), set_necessarily_closed(), and set_not_necessarily_closed().

Referenced by set_necessarily_closed(), and set_not_necessarily_closed().

11.134.3.38 void Parma_Polyhedra_Library::Linear_System::set_sorted (bool *b*) [inline]

Sets the sortedness flag of the system to `b`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 52 of file Linear_System.inlines.hh.

References sorted.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), add_row(), add_rows(), add_rows_and_columns(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), back_substitute(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), gauss(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::minimize(), normalize(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators_with_sat_g(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_constraints(), Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_generators(), resize_no_copy(), sign_normalize(), Parma_Polyhedra_Library::Polyhedron::simplify(), simplify(), sort_and_remove_with_sat(), strong_normalize(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.134.3.39 void Parma_Polyhedra_Library::Linear_System::sign_normalize ()

Sign-normalizes the system.

Definition at line 485 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), and set_sorted().

Referenced by permute_columns(), and Parma_Polyhedra_Library::Polyhedron::simplify().

11.134.3.40 void Parma_Polyhedra_Library::Linear_System::simplify ()

Applies Gaussian elimination and back-substitution so as to simplify the linear system.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, and Parma_Polyhedra_Library::Generator_System.

Definition at line 705 of file Linear_System.cc.

References back_substitute(), Parma_Polyhedra_Library::Matrix::erase_to_end(), gauss(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), set_sorted(), sorted, Parma_Polyhedra_Library::swap(), swap(), and unset_pending_rows().

11.134.3.41 void Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat (Bit_Matrix & *sat*)

Sorts the system, removing duplicates, keeping the saturation matrix consistent.

Parameters

sat Bit matrix with rows corresponding to the rows of `*this`.

Definition at line 515 of file Linear_System.cc.

References check_sorted(), Parma_Polyhedra_Library::Matrix::erase_to_end(), first_pending_row(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Bit_Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::rows, Parma_Polyhedra_Library::Bit_Matrix::rows_erase_to_end(), set_index_first_pending_row(), set_sorted(), and Parma_Polyhedra_Library::swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators(), and Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators_with_sat_g().

11.134.3.42 void Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates ()

Sorts the pending rows and eliminates those that also occur in the non-pending part of the system.

Definition at line 783 of file Linear_System.cc.

References Parma_Polyhedra_Library::cmp(), Parma_Polyhedra_Library::Matrix::erase_to_end(), first_pending_row(), is_sorted(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), sort_rows(), Parma_Polyhedra_Library::swap(), and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::process_pending_constraints(), and Parma_Polyhedra_Library::Polyhedron::process_pending_generators().

11.134.3.43 void Parma_Polyhedra_Library::Linear_System::sort_rows (dimension_type *first_row*, dimension_type *last_row*)

Sorts the rows (in growing order) form `first_row` to `last_row` and eliminates duplicated ones.

Definition at line 329 of file Linear_System.cc.

References first_pending_row(), Parma_Polyhedra_Library::Matrix::num_rows(), and Parma_Polyhedra_Library::Matrix::rows.

11.134.3.44 void Parma_Polyhedra_Library::Linear_System::sort_rows ()

Sorts the non-pending rows (in growing order) and eliminates duplicated ones.

Definition at line 317 of file Linear_System.cc.

References first_pending_row(), num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), set_index_first_pending_row(), and sorted.

Referenced by Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints(), Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators(), Parma_Polyhedra_Library::Polyhedron::OK(), sort_pending_and_remove_duplicates(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.134.3.45 dimension_type Parma_Polyhedra_Library::Linear_System::space_dimension () const [inline]

Returns the space dimension of the rows in the system.

The computation of the space dimension correctly ignores the column encoding the inhomogeneous terms of constraint (resp., the divisors of generators); if the system topology is `NOT_NECESSARILY_CLOSED`, also the column of the ϵ-dimension coefficients will be ignored.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 202 of file Linear_System.inlines.hh.

References is_necessarily_closed(), and Parma_Polyhedra_Library::Matrix::num_columns().

11.134.3.46 void Parma_Polyhedra_Library::Linear_System::strong_normalize ()

Strongly normalizes the system.

Definition at line 475 of file Linear_System.cc.

References Parma_Polyhedra_Library::Matrix::num_rows(), and set_sorted().

Referenced by Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), Parma_Polyhedra_Library::Polyhedron::OK(), OK(), and

remove_trailing_columns().

11.134.3.47 void Parma_Polyhedra_Library::Linear_System::swap (Linear_System & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 132 of file Linear_System.inlines.hh.

References index_first_pending, row_topology, and sorted.

Referenced by add_pending_row(), add_pending_rows(), gauss(), insert(), insert_pending(), merge_rows_assign(), simplify(), sort_pending_and_remove_duplicates(), and swap().

11.134.3.48 Topology Parma_Polyhedra_Library::Linear_System::topology () const [inline]

Returns the system topology.

Definition at line 189 of file Linear_System.inlines.hh.

References row_topology.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::conversion(), insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Polyhedron::OK(), OK(), Parma_Polyhedra_Library::Grid_Generator_System::OK(), Parma_Polyhedra_Library::Polyhedron::select_CH78_constraints(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), and Parma_Polyhedra_Library::Polyhedron::topology().

11.134.3.49 memory_size_type Parma_Polyhedra_Library::Linear_System::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Matrix.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 37 of file Linear_System.inlines.hh.

References external_memory_in_bytes().

11.134.3.50 void Parma_Polyhedra_Library::Linear_System::unset_pending_rows () [inline]

Sets the index to indicate that the system has no pending rows.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 85 of file Linear_System.inlines.hh.

References index_first_pending, and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), add_rows(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Polyhedron::conversion(), Linear_System(), merge_rows_assign(), Parma_Polyhedra_Library::Polyhedron::OK(), operator=(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_constraints(), Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_generators(), Parma_Polyhedra_Library::Polyhedron::simplified_constraints(), Parma_Polyhedra_Library::Polyhedron::simplify(), simplify(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators(), Parma_Polyhedra_Library::Polyhedron::time_elapse_assign(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.134.4 Friends And Related Function Documentation

11.134.4.1 bool operator!= (const Linear_System & *x*, const Linear_System & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 227 of file Linear_System.inlines.hh.

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::const_iterator::operator!=().

11.134.4.2 bool operator== (const Linear_System & *x*, const Linear_System & *y*) [related]

Returns `true` if and only if `x` and `y` are identical.

Definition at line 496 of file Linear_System.cc.

References first_pending_row(), Parma_Polyhedra_Library::Matrix::num_columns(), and Parma_Polyhedra_Library::Matrix::num_rows().

11.134.4.3 void swap (Parma_Polyhedra_Library::Linear_System & *x*, Parma_Polyhedra_Library::Linear_System & *y*) [related]

Specializes `std::swap`.

Definition at line 243 of file Linear_System.inlines.hh.

References swap().

11.134.5 Member Data Documentation

11.134.5.1 dimension_type Parma_Polyhedra_Library::Linear_System::index_first_pending [private]

The index of the first pending row.

Definition at line 384 of file Linear_System.defs.hh.

Referenced by assign_with_pending(), clear(), first_pending_row(), set_index_first_pending_row(), swap(), and unset_pending_rows().

11.134.5.2 Topology Parma_Polyhedra_Library::Linear_System::row_topology [private]

The topological kind of the rows in the system.

Definition at line 381 of file Linear_System.defs.hh.

Referenced by add_pending_rows(), add_rows_and_columns(), assign_with_pending(), is_necessarily_closed(), operator=(), resize_no_copy(), set_necessarily_closed(), set_not_necessarily_closed(), swap(), and topology().

11.134.5.3 bool Parma_Polyhedra_Library::Linear_System::sorted [private]

`true` if rows are sorted in the ascending order as defined by `bool compare(const Linear_Row&, const Linear_Row&)`. If `false` may not be sorted.

Definition at line 391 of file Linear_System.defs.hh.

Referenced by add_pending_rows(), ascii_dump(), assign_with_pending(), clear(), is_sorted(), Linear_System(), OK(), operator=(), set_sorted(), simplify(), sort_rows(), and swap().

The documentation for this class was generated from the following files:

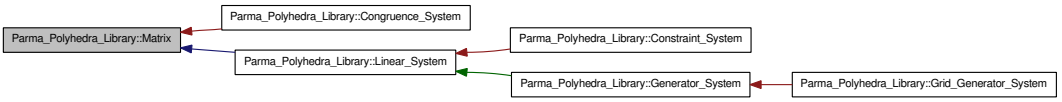
- Linear_System.defs.hh
- Linear_System.inlines.hh
- Linear_System.cc

11.135 Parma_Polyhedra_Library::Matrix Class Reference

A 2-dimensional matrix of coefficients.

`#include <Matrix.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Matrix:



Classes

- class const_iterator

  An iterator over a matrix.

Public Member Functions

- Matrix ()

  Builds an empty matrix.

- Matrix (dimension_type n_rows, dimension_type n_columns, Row::Flags row_flags=Row::Flags())

  Builds a zero matrix with specified dimensions and flags.

- Matrix (const Matrix &y)

  Copy constructor.

- ~Matrix ()

  Destructor.

- Matrix & operator= (const Matrix &y)

  Assignment operator.

- bool has_no_rows () const

  Returns `true` *if and only if* `*this` *has no rows.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first row, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- void swap (Matrix &y)

  Swaps `*this` *with* `y`.

- void add_zero_rows (dimension_type n, Row::Flags row_flags)

  Adds to the matrix `n` *rows of zeroes with flags set to* `row_flags`.

- void add_zero_columns (dimension_type n)

  Adds `n` *columns of zeroes to the matrix.*

- void add_zero_rows_and_columns (dimension_type n, dimension_type m, Row::Flags row_flags)

  Adds `n` rows and `m` columns of zeroes to the matrix.

- void add_row (const Row &y)

  Adds a copy of the row `y` to the matrix.

- void add_recycled_row (Row &y)

  Adds the row `y` to the matrix.

- void remove_trailing_columns (dimension_type n)

  Makes the matrix shrink by removing its `n` trailing columns.

- void resize_no_copy (dimension_type new_n_rows, dimension_type new_n_columns, Row::Flags row_flags)

  Resizes the matrix without worrying about the old contents.

- void swap_columns (dimension_type i, dimension_type j)

  Swaps the columns having indexes `i` and `j`.

- void permute_columns (const std::vector< dimension_type > &cycles)

  Permutes the columns of the matrix.

- void clear ()

  Clears the matrix deallocating all its rows.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- void erase_to_end (dimension_type first_to_erase)

  Erases from the matrix all the rows but those having an index less than `first_to_erase`.

- bool OK () const

  Checks if all the invariants are satisfied.

Accessors

- dimension_type num_columns () const
  Returns the number of columns of the matrix (i.e., the size of the rows).
- dimension_type num_rows () const
  Returns the number of rows in the matrix.

Subscript operators

- Row & operator[ ] (dimension_type k)
  Returns a reference to the `k-th` row of the matrix.
- const Row & operator[ ] (dimension_type k) const
  Returns a constant reference to the `k-th` row of the matrix.

Static Public Member Functions

- static dimension_type max_num_rows ()
  Returns the maximum number of rows of a Matrix.
- static dimension_type max_num_columns ()
  Returns the maximum number of columns of a Matrix.

Protected Attributes

- std::vector< Row > rows
  Contains the rows of the matrix.
- dimension_type row_size
  Size of the initialized part of each row.
- dimension_type row_capacity
  Capacity allocated for each row.

Related Functions

(Note that these are not member functions.)

- void swap (Parma_Polyhedra_Library::Matrix &x, Parma_Polyhedra_Library::Matrix &y)
  Specializes `std::swap`.
- bool operator== (const Matrix &x, const Matrix &y)
  Returns `true` if and only if `x` and `y` are identical.
- bool operator!= (const Matrix &x, const Matrix &y)
  Returns `true` if and only if `x` and `y` are different.

11.135.1 Detailed Description

A 2-dimensional matrix of coefficients. A Matrix object is a sequence of Row objects and is characterized by the matrix dimensions (the number of rows and columns). All the rows in a matrix, besides having the same size (corresponding to the number of columns of the matrix), are also bound to have the same capacity.

Definition at line 46 of file Matrix.defs.hh.

11.135.2 Constructor & Destructor Documentation

11.135.2.1 Parma_Polyhedra_Library::Matrix::Matrix () [inline]

Builds an empty matrix.

Rows' size and capacity are initialized to 0.

Definition at line 122 of file Matrix.inlines.hh.

11.135.2.2 Parma_Polyhedra_Library::Matrix::Matrix (dimension_type *n_rows*, dimension_type *n_columns*, Row::Flags *row_flags* = Row::Flags ())

Builds a zero matrix with specified dimensions and flags.

Parameters

n_rows The number of rows of the matrix that will be created;

n_columns The number of columns of the matrix that will be created.

row_flags The flags used to build the rows of the matrix; by default, the rows will have all flags unset.

Definition at line 33 of file Matrix.cc.

References Parma_Polyhedra_Library::construct(), max_num_rows(), OK(), row_capacity, and rows.

11.135.2.3 Parma_Polyhedra_Library::Matrix::Matrix (const Matrix & *y*) [inline]

Copy constructor.

Definition at line 129 of file Matrix.inlines.hh.

11.135.2.4 Parma_Polyhedra_Library::Matrix::~Matrix () [inline]

Destructor.

Definition at line 136 of file Matrix.inlines.hh.

11.135.3 Member Function Documentation

11.135.3.1 void Parma_Polyhedra_Library::Matrix::add_recycled_row (Row & *y*)

Adds the row `y` to the matrix.

Parameters

y The row to be added: it must have the same size and capacity as `*this`. It is not declared `const` because its data-structures will recycled to build the new matrix row.

Turns the $r \times c$ matrix M into the $(r+1) \times c$ matrix $\binom{M}{y}$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 183 of file Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), max_num_rows(), OK(), Parma_Polyhedra_-Library::Row::OK(), row_capacity, row_size, rows, and swap().

Referenced by add_row(), Parma_Polyhedra_Library::Congruence_System::insert(), and Parma_-Polyhedra_Library::Congruence_System::insert_verbatim().

11.135.3.2 void Parma_Polyhedra_Library::Matrix::add_row (const Row & *y*) [inline]

Adds a copy of the row `y` to the matrix.

Parameters

y The row to be copied: it must have the same number of columns as the matrix.

Turns the $r \times c$ matrix M into the $(r+1) \times c$ matrix $\binom{M}{y}$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 157 of file Matrix.inlines.hh.

References add_recycled_row(), and row_capacity.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_-Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_Library::PIP_Solution_-Node::solve(), and Parma_Polyhedra_Library::PIP_Problem::solve().

11.135.3.3 void Parma_Polyhedra_Library::Matrix::add_zero_columns (dimension_type *n*)

Adds `n` columns of zeroes to the matrix.

Parameters

n The number of columns to be added: must be strictly positive.

Turns the $r \times c$ matrix M into the $r \times (c+n)$ matrix $(M\ 0)$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 83 of file Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), max_num_columns(), num_columns(), num_rows(), row_capacity, row_size, rows, and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Grid::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Grid_Generator_System::clear(), Parma_Polyhedra_Library::Congruence_System::clear(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_Library::MIP_Problem::is_lp_satisfiable(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), and Parma_Polyhedra_Library::PIP_Problem::solve().

11.135.3.4 void Parma_Polyhedra_Library::Matrix::add_zero_rows (dimension_type *n*, Row::Flags *row_flags*)

Adds to the matrix `n` rows of zeroes with flags set to `row_flags`.

Parameters

n The number of rows to be added: must be strictly positive.

row_flags Flags for the newly added rows.

Turns the $r \times c$ matrix M into the $(r+n) \times c$ matrix $\binom{M}{0}$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 52 of file Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::construct(), max_num_rows(), num_rows(), row_capacity, row_size, rows, and swap().

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_rows(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), and Parma_Polyhedra_Library::Grid::simplify().

11.135.3.5 void Parma_Polyhedra_Library::Matrix::add_zero_rows_and_columns (dimension_type *n*, dimension_type *m*, Row::Flags *row_flags*)

Adds `n` rows and `m` columns of zeroes to the matrix.

Parameters

n The number of rows to be added: must be strictly positive.

m The number of columns to be added: must be strictly positive.

row_flags Flags for the newly added rows.

Turns the $r \times c$ matrix M into the $(r+n) \times (c+m)$ matrix $\left(\begin{smallmatrix} M & 0 \\ 0 & 0 \end{smallmatrix}\right)$. The matrix is expanded avoiding reallocation whenever possible.

Definition at line 110 of file Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), max_num_columns(), max_num_rows(), num_columns(), num_rows(), row_capacity, row_size, rows, and swap().

Referenced by Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), Parma_Polyhedra_Library::Grid_Generator_System::add_universe_rows_and_columns(), Parma_Polyhedra_Library::Congruence_System::concatenate(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), and Parma_Polyhedra_Library::Congruence_System::recycling_insert().

11.135.3.6 void Parma_Polyhedra_Library::Matrix::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 292 of file Matrix.cc.

References ascii_dump(), num_columns(), and num_rows().

11.135.3.7 void Parma_Polyhedra_Library::Matrix::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::ascii_dump(), Parma_Polyhedra_Library::PIP_Problem::ascii_dump(), Parma_Polyhedra_Library::MIP_Problem::ascii_dump(), ascii_dump(), and Parma_Polyhedra_Library::PIP_Solution_Node::solve().

11.135.3.8 bool Parma_Polyhedra_Library::Matrix::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System,

and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 304 of file Matrix.cc.

References Parma_Polyhedra_Library::ascii_load().

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::ascii_load().

11.135.3.9 Matrix::const_iterator Parma_Polyhedra_Library::Matrix::begin () const [inline]

Returns the const_iterator pointing to the first row, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 105 of file Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Constraint_System::begin(), and Parma_Polyhedra_Library::Congruence_System::begin().

11.135.3.10 void Parma_Polyhedra_Library::Matrix::clear () [inline]

Clears the matrix deallocating all its rows.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 198 of file Matrix.inlines.hh.

References row_capacity, row_size, rows, and swap().

Referenced by Parma_Polyhedra_Library::PIP_Problem::clear().

11.135.3.11 Matrix::const_iterator Parma_Polyhedra_Library::Matrix::end () const [inline]

Returns the past-the-end const_iterator.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 110 of file Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Generator_System::const_iterator::skip_forward(), Parma_Polyhedra_Library::Constraint_System::const_iterator::skip_forward(), and Parma_Polyhedra_Library::Congruence_System::const_iterator::skip_forward().

11.135.3.12 void Parma_Polyhedra_Library::Matrix::erase_to_end (dimension_type *first_to_erase*) [inline]

Erases from the matrix all the rows but those having an index less than `first_to_erase`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 191 of file Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::MIP_Problem::erase_artificials(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), Parma_Polyhedra_Library::Polyhedron::simplify(), Parma_Polyhedra_Library::Linear_System::simplify(), Parma_Polyhedra_Library::Grid::simplify(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators(), and Parma_Polyhedra_Library::Polyhedron::time_elapse_assign().

11.135.3.13 PPL::memory_size_type Parma_Polyhedra_Library::Matrix::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 388 of file Matrix.cc.

References num_rows(), row_capacity, and rows.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::external_memory_in_bytes(), Parma_Polyhedra_Library::PIP_Problem::external_memory_in_bytes(), Parma_Polyhedra_Library::MIP_Problem::external_memory_in_bytes(), and total_memory_in_bytes().

11.135.3.14 bool Parma_Polyhedra_Library::Matrix::has_no_rows () const [inline]

Returns `true` if and only if `*this` has no rows.

Note

The unusual naming for this method is *intentional:* we do not want it to be named `empty` because this would cause an error prone name clash with the corresponding methods in derived classes Constraint_System and Congruence_System (which have a different semantics).

Definition at line 100 of file Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Grid::add_recycled_congruences(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Grid::add_recycled_grid_generators(), Parma_Polyhedra_Library::Linear_System::add_rows(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_widening_assign(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Generator_System::empty(), Parma_Polyhedra_Library::Grid::generator_widening_assign(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Grid::grid_generators(), Parma_Polyhedra_Library::Polyhedron::H79_widening_assign(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_Library::Grid::intersection_assign(), Parma_Polyhedra_Library::Polyhedron::map_space_dimensions(), Parma_Polyhedra_Library::Grid::map_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::Grid::minimized_grid_generators(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Linear_System::OK(), Parma_Polyhedra_Library::Grid::OK(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Linear_System::set_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), Parma_Polyhedra_Library::Grid::simplify(), and Parma_Polyhedra_Library::Grid::update_congruences().

11.135.3.15 dimension_type Parma_Polyhedra_Library::Matrix::max_num_columns () [inline, static]

Returns the maximum number of columns of a Matrix.

Definition at line 38 of file Matrix.inlines.hh.

References Parma_Polyhedra_Library::Row::max_size().

Referenced by add_zero_columns(), add_zero_rows_and_columns(), Parma_Polyhedra_Library::Linear_System::max_space_dimension(), Parma_Polyhedra_Library::Congruence_System::max_space_dimension(), operator=(), and resize_no_copy().

11.135.3.16 dimension_type Parma_Polyhedra_Library::Matrix::max_num_rows () [inline, static]

Returns the maximum number of rows of a Matrix.

Definition at line 33 of file Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), add_recycled_row(), add_zero_rows(), add_zero_rows_and_columns(), Matrix(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), and resize_no_copy().

11.135.3.17 dimension_type Parma_Polyhedra_Library::Matrix::num_columns () const [inline]

Returns the number of columns of the matrix (i.e., the size of the rows).

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 180 of file Matrix.inlines.hh.

References row_size.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Generator_System::add_corresponding_points(), Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_generators(), Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Grid::add_space_dimensions(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), add_zero_columns(), add_zero_rows_and_columns(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), ascii_dump(), Parma_Polyhedra_Library::Linear_System::ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), Parma_Polyhedra_Library::Congruence_System::ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_load(), Parma_Polyhedra_Library::Constraint_System::ascii_load(), Parma_Polyhedra_Library::Linear_System::back_substitute(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_exact_pricing(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::Congruence_System::concatenate(), Parma_Polyhedra_Library::Polyhedron::concatenate_assign(), Parma_Polyhedra_Library::Grid::congruences(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::MIP_Problem::erase_artificials(), Parma_Polyhedra_Library::Linear_System::gauss(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::Congruence_System::has_linear_equalities(), Parma_Polyhedra_Library::Generator_System::has_points(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot(), Parma_Polyhedra_Library::Congruence_System::is_equal_to(), Parma_Polyhedra_Library::MIP_Problem::is_lp_satisfiable(), Parma_Polyhedra_Library::Polyhedron::is_universe(), Parma_Polyhedra_Library::Grid::lower_triangular(), Parma_Polyhedra_Library::MIP_Problem::merge_split_variable(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::MIP_Problem::OK(), Parma_Polyhedra_Library::Linear_System::OK(), Parma_Polyhedra_Library::Grid::OK(), Parma_Polyhedra_Library::Congruence_System::OK(), operator==(), Parma_Polyhedra_Library::Linear_System::operator==(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::scale(), Parma_Polyhedra_Library::Polyhedron::simplify(), Parma_Polyhedra_Library::Grid::simplify(), Parma_Polyhedra_Library::PIP_Problem::solve(), Parma_

Polyhedra_Library::Linear_System::space_dimension(), Parma_Polyhedra_Library::Congruence_-System::space_dimension(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_-entering_index(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_float_entering_index(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints(), Parma_Polyhedra_-Library::Polyhedron::strongly_minimize_generators(), and swap_columns().

11.135.3.18 dimension_type Parma_Polyhedra_Library::Matrix::num_rows () const [inline]

Returns the number of rows in the matrix.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 175 of file Matrix.inlines.hh.

References rows.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_and_minimize(), Parma_-Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_-Library::Generator_System::add_corresponding_points(), Parma_Polyhedra_Library::Polyhedron::add_-generator(), Parma_Polyhedra_Library::Linear_System::add_pending_rows(), Parma_Polyhedra_-Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::add_recycled_-generators(), Parma_Polyhedra_Library::Linear_System::add_row(), Parma_Polyhedra_Library::Linear_-System::add_rows(), Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_-Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::add_-space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_-project(), Parma_Polyhedra_Library::Congruence_System::add_unit_rows_and_columns(), add_zero_-columns(), add_zero_rows(), add_zero_rows_and_columns(), Parma_Polyhedra_Library::Generator_-System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_-System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Grid::affine_dimension(), Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Constraint_-System::affine_preimage(), Parma_Polyhedra_Library::Congruence_System::affine_preimage(), ascii_dump(), Parma_Polyhedra_Library::Linear_System::ascii_dump(), Parma_Polyhedra_-Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_-dump(), Parma_Polyhedra_Library::Congruence_System::ascii_dump(), Parma_Polyhedra_-Library::Generator_System::ascii_load(), Parma_Polyhedra_Library::Constraint_System::ascii_load(), Parma_Polyhedra_Library::Congruence_System::ascii_load(), Parma_Polyhedra_Library::Linear_-System::back_substitute(), Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_-exact(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_-Library::Polyhedron::BHRZ03_evolving_points(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_-evolving_rays(), Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_-Library::Polyhedron::bounds(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_exact_pricing(), Parma_-Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_-Library::Congruence_System::concatenate(), Parma_Polyhedra_Library::Polyhedron::concatenate_-assign(), Parma_Polyhedra_Library::Grid::congruence_widening_assign(), Parma_Polyhedra_-Library::Grid::congruences(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_-Library::Grid::constrains(), Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_-Library::Grid::construct(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_-Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::drop_some_-non_integer_points(), Parma_Polyhedra_Library::MIP_Problem::erase_artificials(), external_-memory_in_bytes(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_-Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::PIP_Solution_-

Node::generate_cut(), Parma_Polyhedra_Library::Polyhedron::generators(), Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index(), Parma_Polyhedra_Library::Polyhedron::H79_widening_assign(), Parma_Polyhedra_Library::Congruence_System::has_a_free_dimension(), Parma_Polyhedra_Library::Constraint_System::has_equalities(), Parma_Polyhedra_Library::Congruence_System::has_linear_equalities(), Parma_Polyhedra_Library::Generator_System::has_points(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot(), Parma_Polyhedra_Library::Polyhedron::is_bounded(), Parma_Polyhedra_Library::Congruence_System::is_equal_to(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), Parma_Polyhedra_Library::Polyhedron::is_topologically_closed(), Parma_Polyhedra_Library::Polyhedron::is_universe(), Parma_Polyhedra_Library::Grid::is_universe(), Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_congruence_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_generator_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign(), Parma_Polyhedra_Library::Grid::lower_triangular(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), Parma_Polyhedra_Library::Polyhedron::minimize(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize(), Parma_Polyhedra_Library::Linear_System::normalize(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Congruence_System::num_equalities(), Parma_Polyhedra_Library::Constraint_System::num_inequalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Linear_System::num_lines_or_equalities(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Congruence_System::num_proper_congruences(), Parma_Polyhedra_Library::Generator_System::num_rays(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::OK(), Parma_Polyhedra_Library::MIP_Problem::OK(), OK(), Parma_Polyhedra_Library::Linear_System::OK(), Parma_Polyhedra_Library::Generator_System::OK(), Parma_Polyhedra_Library::Constraint_System::OK(), Parma_Polyhedra_Library::Congruence_System::OK(), operator==(), Parma_Polyhedra_Library::Linear_System::operator==(), permute_columns(), Parma_Polyhedra_Library::MIP_Problem::pivot(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints(), Parma_Polyhedra_Library::Polyhedron::quick_equivalence_test(), Parma_Polyhedra_Library::Grid::quick_equivalence_test(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Grid::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), remove_trailing_columns(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::scale(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), Parma_Polyhedra_Library::Polyhedron::select_CH78_constraints(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), Parma_Polyhedra_Library::Linear_System::set_rows_topology(), Parma_Polyhedra_Library::Linear_System::sign_normalize(), Parma_Polyhedra_Library::Polyhedron::simplify(), Parma_Polyhedra_Library::Linear_System::simplify(), Parma_Polyhedra_Library::Grid::simplify(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), Parma_Polyhedra_Library::Grid::simplify_using_context_assign(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_Decision_Node::solve(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), Parma_Polyhedra_Library::Linear_System::sort_rows(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_float_entering_index(), Parma_Polyhedra_

Library::Linear_System::strong_normalize(), Parma_Polyhedra_Library::Polyhedron::strongly_-minimize_constraints(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators(), swap_columns(), Parma_Polyhedra_Library::Polyhedron::time_elapse_assign(), Parma_Polyhedra_-Library::Polyhedron::topological_closure_assign(), and Parma_Polyhedra_Library::Linear_-System::unset_pending_rows().

11.135.3.19 bool Parma_Polyhedra_Library::Matrix::OK () const

Checks if all the invariants are satisfied.

Reimplemented in Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_-Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_-Library::Grid_Generator_System.

Definition at line 396 of file Matrix.cc.

References num_rows(), row_capacity, and row_size.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_-Library::Linear_System::add_pending_rows(), add_recycled_row(), Parma_Polyhedra_Library::Linear_-System::add_row(), Parma_Polyhedra_Library::Linear_System::add_rows(), Parma_Polyhedra_-Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::PIP_Tree_-Node::compatibility_check(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_-Library::Linear_System::insert_pending(), Matrix(), Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau::OK(), Parma_Polyhedra_Library::PIP_Problem::OK(), Parma_Polyhedra_Library::MIP_-Problem::OK(), Parma_Polyhedra_Library::Linear_System::OK(), Parma_Polyhedra_Library::Grid_-Generator_System::OK(), Parma_Polyhedra_Library::Generator_System::OK(), Parma_Polyhedra_-Library::Constraint_System::OK(), Parma_Polyhedra_Library::Congruence_System::OK(), Parma_-Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), and Parma_Polyhedra_-Library::Linear_System::sort_rows().

11.135.3.20 Matrix & Parma_Polyhedra_Library::Matrix::operator= (const Matrix & *y*) [inline]

Assignment operator.

Definition at line 140 of file Matrix.inlines.hh.

References Parma_Polyhedra_Library::compute_capacity(), max_num_columns(), row_capacity, row_-size, and rows.

11.135.3.21 const Row & Parma_Polyhedra_Library::Matrix::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the `k-th` row of the matrix.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_-System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 169 of file Matrix.inlines.hh.

References rows.

11.135.3.22 Row & Parma_Polyhedra_Library::Matrix::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the `k-th` row of the matrix.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 163 of file Matrix.inlines.hh.

References rows.

11.135.3.23 void Parma_Polyhedra_Library::Matrix::permute_columns (const std::vector< dimension_type > & *cycles*)

Permutes the columns of the matrix.

Reimplemented in Parma_Polyhedra_Library::Linear_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 346 of file Matrix.cc.

References num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, rows, Parma_Polyhedra_Library::swap(), and swap().

Referenced by Parma_Polyhedra_Library::Grid::map_space_dimensions(), and Parma_Polyhedra_Library::MIP_Problem::merge_split_variable().

11.135.3.24 void Parma_Polyhedra_Library::Matrix::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

11.135.3.25 void Parma_Polyhedra_Library::Matrix::remove_trailing_columns (dimension_type *n*)

Makes the matrix shrink by removing its `n` trailing columns.

Reimplemented in Parma_Polyhedra_Library::Linear_System.

Definition at line 337 of file Matrix.cc.

References num_rows(), row_size, and rows.

Referenced by Parma_Polyhedra_Library::MIP_Problem::erase_artificials(), Parma_Polyhedra_-Library::MIP_Problem::merge_split_variable(), and Parma_Polyhedra_Library::Congruence_-System::remove_higher_space_dimensions().

11.135.3.26 void Parma_Polyhedra_Library::Matrix::resize_no_copy (dimension_type *new_n_rows*, dimension_type *new_n_columns*, Row::Flags *row_flags*)

Resizes the matrix without worrying about the old contents.

Parameters

new_n_rows The number of rows of the resized matrix;

new_n_columns The number of columns of the resized matrix.

row_flags The flags of the rows eventually added to the matrix.

The matrix is expanded to the specified dimensions avoiding reallocation whenever possible. The contents of the original matrix is lost.

Definition at line 212 of file Matrix.cc.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::construct(), max_num_columns(), max_num_rows(), row_capacity, row_size, rows, and swap().

11.135.3.27 void Parma_Polyhedra_Library::Matrix::swap (Matrix & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 115 of file Matrix.inlines.hh.

References row_capacity, row_size, and rows.

Referenced by add_recycled_row(), add_zero_columns(), add_zero_rows(), add_zero_rows_and_-columns(), clear(), permute_columns(), resize_no_copy(), swap(), and swap_columns().

11.135.3.28 void Parma_Polyhedra_Library::Matrix::swap_columns (dimension_type *i*, dimension_type *j*)

Swaps the columns having indexes `i` and `j`.

Definition at line 328 of file Matrix.cc.

References num_columns(), num_rows(), rows, and swap().

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions(), Parma_Polyhedra_-Library::Grid::add_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::add_space_-dimensions_and_embed(), Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), Parma_Polyhedra_-Library::Congruence_System::add_unit_rows_and_columns(), Parma_Polyhedra_Library::Grid_-Generator_System::add_universe_rows_and_columns(), Parma_Polyhedra_Library::Generator_-System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_-System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Polyhedron::concatenate_-

assign(), Parma_Polyhedra_Library::Congruence_System::increase_space_dimension(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_Library::Grid_Generator_System::recycling_insert(), Parma_Polyhedra_Library::Congruence_System::recycling_insert(), Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Grid_Generator_System::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Congruence_System::remove_higher_space_dimensions(), and Parma_Polyhedra_Library::Grid_Generator_System::remove_space_dimensions().

11.135.3.29 memory_size_type Parma_Polyhedra_Library::Matrix::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_System, Parma_Polyhedra_Library::Constraint_System, Parma_Polyhedra_Library::Congruence_System, Parma_Polyhedra_Library::Generator_System, and Parma_Polyhedra_Library::Grid_Generator_System.

Definition at line 43 of file Matrix.inlines.hh.

References external_memory_in_bytes().

11.135.4 Friends And Related Function Documentation

11.135.4.1 bool operator!= (const Matrix & *x*, const Matrix & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 186 of file Matrix.inlines.hh.

11.135.4.2 bool operator== (const Matrix & *x*, const Matrix & *y*) [related]

Returns `true` if and only if `x` and `y` are identical.

Definition at line 374 of file Matrix.cc.

References num_columns(), and num_rows().

11.135.4.3 void swap (Parma_Polyhedra_Library::Matrix & *x*, Parma_Polyhedra_Library::Matrix & *y*) [related]

Specializes `std::swap`.

Definition at line 211 of file Matrix.inlines.hh.

References swap().

11.135.5 Member Data Documentation

11.135.5.1 dimension_type Parma_Polyhedra_Library::Matrix::row_capacity [protected]

Capacity allocated for each row.

Definition at line 174 of file Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Linear_System::add_pending_rows(), add_recycled_row(), add_row(), add_zero_columns(), add_zero_rows(), add_zero_rows_and_columns(), clear(), external_memory_in_bytes(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Grid_Generator_System::insert(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_pending(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Matrix(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), OK(), Parma_Polyhedra_Library::Linear_System::OK(), operator=(), resize_no_copy(), and swap().

11.135.5.2 dimension_type Parma_Polyhedra_Library::Matrix::row_size [protected]

Size of the initialized part of each row.

Definition at line 171 of file Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Linear_System::add_pending_rows(), add_recycled_row(), Parma_Polyhedra_Library::Linear_System::add_row(), add_zero_columns(), add_zero_rows(), add_zero_rows_and_columns(), clear(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), num_columns(), OK(), Parma_Polyhedra_Library::Linear_System::OK(), operator=(), remove_trailing_columns(), resize_no_copy(), and swap().

11.135.5.3 std::vector<Row> Parma_Polyhedra_Library::Matrix::rows [protected]

Contains the rows of the matrix.

Definition at line 168 of file Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::Linear_System::add_pending_row(), add_recycled_row(), add_zero_columns(), add_zero_rows(), add_zero_rows_and_columns(), begin(), clear(), end(), erase_to_end(), external_memory_in_bytes(), has_no_rows(), Parma_Polyhedra_Library::Congruence_System::insert(), Matrix(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), num_rows(), operator=(), operator[ ](), permute_columns(), remove_trailing_columns(), resize_no_copy(), Parma_Polyhedra_Library::Grid::simplify(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_rows(), swap(), and swap_columns().

The documentation for this class was generated from the following files:

- Matrix.defs.hh
- Matrix.inlines.hh
- Matrix.cc

11.136 Parma_Polyhedra_Library::maybe_assign_struct< To, From > Struct Template Reference

`#include <distances.inlines.hh>`

Static Public Member Functions

- static Result function (const To *&top, To &tmp, const From &from, Rounding_Dir dir)

11.136.1 Detailed Description

template<typename To, typename From> struct Parma_Polyhedra_Library::maybe_assign_struct< To, From >

Definition at line 34 of file distances.inlines.hh.

11.136.2 Member Function Documentation

11.136.2.1 template<typename To , typename From > static Result Parma_Polyhedra_Library::maybe_assign_struct< To, From >::function (const To *& *top*, To & *tmp*, const From & *from*, Rounding_Dir *dir*) [inline, static]

Definition at line 36 of file distances.inlines.hh.

References Parma_Polyhedra_Library::assign_r().

Referenced by Parma_Polyhedra_Library::maybe_assign().

The documentation for this struct was generated from the following file:

- distances.inlines.hh

11.137 Parma_Polyhedra_Library::maybe_assign_struct< Type, Type > Struct Template Reference

`#include <distances.inlines.hh>`

Static Public Member Functions

- static Result function (const Type *&top, Type &, const Type &from, Rounding_Dir)

11.137.1 Detailed Description

template<typename Type> struct Parma_Polyhedra_Library::maybe_assign_struct< Type, Type >

Definition at line 45 of file distances.inlines.hh.

11.137.2 Member Function Documentation

11.137.2.1 template<typename Type > static Result Parma_Polyhedra_Library::maybe_assign_-struct< Type, Type >::function (const Type *& *top*, Type &, const Type & *from*, Rounding_Dir) [inline, static]

Definition at line 47 of file distances.inlines.hh.

The documentation for this struct was generated from the following file:

- distances.inlines.hh

11.138 Parma_Polyhedra_Library::Minus_Infinity Struct Reference

`#include <checked.defs.hh>`

Static Public Attributes

- static const Result_Class vclass = VC_MINUS_INFINITY

11.138.1 Detailed Description

Definition at line 540 of file checked.defs.hh.

11.138.2 Member Data Documentation

11.138.2.1 const Result_Class Parma_Polyhedra_Library::Minus_Infinity::vclass = VC_MINUS_INFINITY [static]

Definition at line 541 of file checked.defs.hh.

The documentation for this struct was generated from the following file:

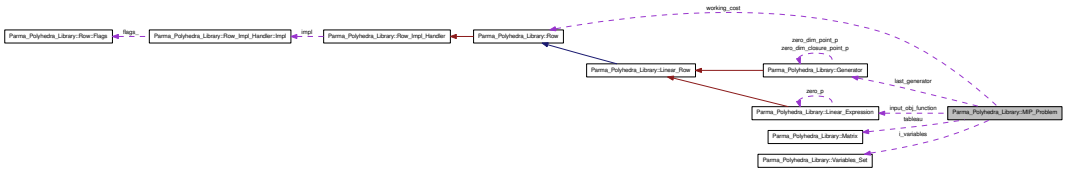
- checked.defs.hh

11.139 Parma_Polyhedra_Library::MIP_Problem Class Reference

A Mixed Integer (linear) Programming problem.

`#include <MIP_Problem.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::MIP_Problem:



Classes

- struct RAII_Temporary_Real_Relaxation

  A helper class to temporarily relax a MIP problem using RAII.

Public Types

- enum Control_Parameter_Name { PRICING }

  Names of MIP problems' control parameters.

- enum Control_Parameter_Value { PRICING_STEEPEST_EDGE_FLOAT, PRICING_STEEPEST_EDGE_EXACT, PRICING_TEXTBOOK }

  Possible values for MIP problem's control parameters.

- typedef Constraint_Sequence::const_iterator const_iterator

  A type alias for the read-only iterator on the constraints defining the feasible region.

Public Member Functions

- MIP_Problem (dimension_type dim=0)

  Builds a trivial MIP problem.

- template<typename In >
  MIP_Problem (dimension_type dim, In first, In last, const Variables_Set &int_vars, const Linear_Expression &obj=Linear_Expression::zero(), Optimization_Mode mode=MAXIMIZATION)

  Builds an MIP problem having space dimension `dim` *from the sequence of constraints in the range* $[\mathrm{first}, \mathrm{last})$*, the objective function* `obj` *and optimization mode* `mode`*; those dimensions whose indices occur in* `int_vars` *are constrained to take an integer value.*

- template<typename In >
  MIP_Problem (dimension_type dim, In first, In last, const Linear_Expression &obj=Linear_Expression::zero(), Optimization_Mode mode=MAXIMIZATION)

  Builds an MIP problem having space dimension `dim` *from the sequence of constraints in the range* $[\mathrm{first}, \mathrm{last})$*, the objective function* `obj` *and optimization mode* `mode`*.*

- MIP_Problem (dimension_type dim, const Constraint_System &cs, const Linear_Expression &obj=Linear_Expression::zero(), Optimization_Mode mode=MAXIMIZATION)

  Builds an MIP problem having space dimension `dim` *from the constraint system* `cs`*, the objective function* `obj` *and optimization mode* `mode`*.*

- MIP_Problem (const MIP_Problem &y)

  Ordinary copy constructor.

- ~MIP_Problem ()

  Destructor.

- MIP_Problem & operator= (const MIP_Problem &y)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the space dimension of the MIP problem.

- const Variables_Set & integer_space_dimensions () const

  Returns a set containing all the variables' indexes constrained to be integral.

- const_iterator constraints_begin () const

  Returns a read-only iterator to the first constraint defining the feasible region.

- const_iterator constraints_end () const

  Returns a past-the-end read-only iterator to the sequence of constraints defining the feasible region.

- const Linear_Expression & objective_function () const

  Returns the objective function.

- Optimization_Mode optimization_mode () const

  Returns the optimization mode.

- void clear ()

  Resets `*this` *to be equal to the trivial MIP problem.*

- void add_space_dimensions_and_embed (dimension_type m)

  Adds `m` *new space dimensions and embeds the old MIP problem in the new vector space.*

- void add_to_integer_space_dimensions (const Variables_Set &i_vars)

  Sets the variables whose indexes are in set `i_vars` *to be integer space dimensions.*

- void add_constraint (const Constraint &c)

  Adds a copy of constraint `c` *to the MIP problem.*

- void add_constraints (const Constraint_System &cs)

  Adds a copy of the constraints in `cs` *to the MIP problem.*

- void set_objective_function (const Linear_Expression &obj)

  Sets the objective function to `obj`*.*

- void set_optimization_mode (Optimization_Mode mode)

  Sets the optimization mode to `mode`*.*

- bool is_satisfiable () const

  Checks satisfiability of `*this`*.*

- MIP_Problem_Status solve () const

  Optimizes the MIP problem.

- void evaluate_objective_function (const Generator &evaluating_point, Coefficient &num, Coefficient &den) const

  Sets `num` *and* `den` *so that* $\frac{num}{den}$ *is the result of evaluating the objective function on* `evaluating_point`*.*

- const Generator & feasible_point () const

Returns a feasible point for `*this`*, if it exists.*

- const Generator & optimizing_point () const

  Returns an optimal point for `*this`*, if it exists.*

- void optimal_value (Coefficient &num, Coefficient &den) const

  Sets `num` *and* `den` *so that* $\frac{num}{den}$ *is the solution of the optimization problem.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`*.*

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`*.*

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`*.*

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`*.*

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`*.*

- void swap (MIP_Problem &y)

  Swaps `*this` *with* `y`*.*

- Control_Parameter_Value get_control_parameter (Control_Parameter_Name name) const

  Returns the value of the control parameter `name`*.*

- void set_control_parameter (Control_Parameter_Value value)

  Sets control parameter `value`*.*

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension an MIP_Problem can handle.

Private Types

- enum Status {
  UNSATISFIABLE, SATISFIABLE, UNBOUNDED, OPTIMIZED,
  PARTIALLY_SATISFIABLE }

  An enumerated type describing the internal status of the MIP problem.

- typedef std::vector< Constraint > Constraint_Sequence

  A type alias for a sequence of constraints.

Private Member Functions

- bool process_pending_constraints ()

  Processes the pending constraints of `*this`.

- void second_phase ()

  Optimizes the MIP problem using the second phase of the primal simplex algorithm.

- MIP_Problem_Status compute_tableau (std::vector< dimension_type > &worked_out_row)

  Assigns to `this->tableau` *a simplex tableau representing the MIP problem, inserting into* `this->mapping` *the information that is required to recover the original MIP problem.*

- bool parse_constraints (dimension_type &additional_tableau_rows, dimension_type &additional_slack_variables, std::deque< bool > &is_tableau_constraint, std::deque< bool > &is_satisfied_inequality, std::deque< bool > &is_nonnegative_variable, std::deque< bool > &is_remergeable_variable) const

  Parses the pending constraints to gather information on how to resize the tableau.

- dimension_type get_exiting_base_index (dimension_type entering_var_index) const

  Computes the row index of the variable exiting the base of the MIP problem. Implemented with anti-cycling rule.

- void pivot (dimension_type entering_var_index, dimension_type exiting_base_index)

  Performs the pivoting operation on the tableau.

- dimension_type textbook_entering_index () const

  Computes the column index of the variable entering the base, using the textbook algorithm with anti-cycling rule.

- dimension_type steepest_edge_exact_entering_index () const

  Computes the column index of the variable entering the base, using an exact steepest-edge algorithm with anti-cycling rule.

- dimension_type steepest_edge_float_entering_index () const

  Same as steepest_edge_exact_entering_index, but using floating points.

- bool compute_simplex_using_exact_pricing ()

  Returns `true` *if and if only the algorithm successfully computed a feasible solution.*

- bool compute_simplex_using_steepest_edge_float ()

  Returns `true` *if and if only the algorithm successfully computed a feasible solution.*

- void erase_artificials (dimension_type begin_artificials, dimension_type end_artificials)

  Drop unnecessary artificial variables from the tableau and get ready for the second phase of the simplex algorithm.

- bool is_in_base (dimension_type var_index, dimension_type &row_index) const
- void compute_generator () const

  Computes a valid generator that satisfies all the constraints of the Linear Programming problem associated to `*this`.

- dimension_type merge_split_variable (dimension_type var_index)

  Merges back the positive and negative part of a (previously split) variable after detecting a corresponding nonnegativity constraint.

- bool is_lp_satisfiable () const

  Returns `true` *if and if only the LP problem is satisfiable.*

Static Private Member Functions

- static void linear_combine (Row &x, const Row &y, const dimension_type k)

  Linearly combines `x` *with* `y` *so that* `*this[k]` *is 0.*

- static bool is_satisfied (const Constraint &c, const Generator &g)

  Returns `true` *if and only if* `c` *is satisfied by* `g`.

- static bool is_saturated (const Constraint &c, const Generator &g)

  Returns `true` *if and only if* `c` *is saturated by* `g`.

- static MIP_Problem_Status solve_mip (bool &have_incumbent_solution, mpq_class &incumbent_solution_value, Generator &incumbent_solution_point, MIP_Problem &mip, const Variables_Set &i_vars)

  Returns a status that encodes the solution of the MIP problem.

- static bool is_mip_satisfiable (MIP_Problem &lp, const Variables_Set &i_vars, Generator &p)

  Returns `true` *if and if only the LP problem* `lp` *is satisfiable when variables in* `i_vars` *can only take integral values.*

- static bool choose_branching_variable (const MIP_Problem &lp, const Variables_Set &i_vars, dimension_type &branching_index)

  Returns `true` *if and if only* `mip.last_generator` *satisfies all the integrality coditions implicitly stated using by* `i_vars`.

Private Attributes

- dimension_type external_space_dim

  The dimension of the vector space.

- dimension_type internal_space_dim

  The space dimension of the current (partial) solution of the MIP problem; it may be smaller than `external_space_dim`.

- Matrix tableau

  The matrix encoding the current feasible region in tableau form.

- Row working_cost

  The working cost function.

- std::vector< std::pair< dimension_type, dimension_type > > mapping

  A map between the variables of 'input_cs' and 'tableau'.

- std::vector< dimension_type > base

  The current basic solution.

- Status status

  The internal state of the MIP problem.

- Control_Parameter_Value pricing

  The pricing method in use.

- bool initialized

  A Boolean encoding whether or not internal data structures have already been properly sized and populated: useful to allow for deeper checks in method OK().

- Constraint_Sequence input_cs

  The sequence of constraints describing the feasible region.

- dimension_type first_pending_constraint

  The first index of 'input_cs' containing a pending constraint.

- Linear_Expression input_obj_function

  The objective function to be optimized.

- Optimization_Mode opt_mode

  The optimization mode requested.

- Generator last_generator

  The last successfully computed feasible or optimizing point.

- Variables_Set i_variables

  A set containing all the indexes of variables that are constrained to have an integer value.

Friends

- class RAII_Temporary_Real_Relaxation

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const MIP_Problem &lp)

  Output operator.

- void swap (Parma_Polyhedra_Library::MIP_Problem &x, Parma_Polyhedra_Library::MIP_Problem &y)

  Specializes `std::swap`.

11.139.1 Detailed Description

A Mixed Integer (linear) Programming problem. An object of this class encodes a mixed integer (linear) programming problem. The MIP problem is specified by providing:

- the dimension of the vector space;
- the feasible region, by means of a finite set of linear equality and non-strict inequality constraints;
- the subset of the unknown variables that range over the integers (the other variables implicitly ranging over the reals);
- the objective function, described by a Linear_Expression;
- the optimization mode (either maximization or minimization).

The class provides support for the (incremental) solution of the MIP problem based on variations of the revised simplex method and on branch-and-bound techniques. The result of the resolution process is expressed in terms of an enumeration, encoding the feasibility and the unboundedness of the optimization problem. The class supports simple feasibility tests (i.e., no optimization), as well as the extraction of an optimal (resp., feasible) point, provided the MIP_Problem is optimizable (resp., feasible).

By exploiting the incremental nature of the solver, it is possible to reuse part of the computational work already done when solving variants of a given MIP_Problem: currently, incremental resolution supports the addition of space dimensions, the addition of constraints, the change of objective function and the change of optimization mode.

Definition at line 80 of file MIP_Problem.defs.hh.

11.139.2 Member Typedef Documentation

11.139.2.1 typedef Constraint_Sequence::const_iterator Parma_Polyhedra_Library::MIP_Problem::const_iterator

A type alias for the read-only iterator on the constraints defining the feasible region.

Definition at line 237 of file MIP_Problem.defs.hh.

11.139.2.2 typedef std::vector<Constraint> Parma_Polyhedra_Library::MIP_Problem::Constraint_Sequence [private]

A type alias for a sequence of constraints.

Definition at line 230 of file MIP_Problem.defs.hh.

11.139.3 Member Enumeration Documentation

11.139.3.1 enum Parma_Polyhedra_Library::MIP_Problem::Control_Parameter_Name

Names of MIP problems' control parameters.

Enumerator:

PRICING The pricing rule.

Definition at line 402 of file MIP_Problem.defs.hh.

11.139.3.2 enum Parma_Polyhedra_Library::MIP_Problem::Control_Parameter_Value

Possible values for MIP problem's control parameters.

Enumerator:

PRICING_STEEPEST_EDGE_FLOAT Steepest edge pricing method, using floating points (default).

PRICING_STEEPEST_EDGE_EXACT Steepest edge pricing method, using Coefficient.

PRICING_TEXTBOOK Textbook pricing method.

Definition at line 408 of file MIP_Problem.defs.hh.

11.139.3.3 enum Parma_Polyhedra_Library::MIP_Problem::Status [private]

An enumerated type describing the internal status of the MIP problem.

Enumerator:

UNSATISFIABLE The MIP problem is unsatisfiable.

SATISFIABLE The MIP problem is satisfiable; a feasible solution has been computed.

UNBOUNDED The MIP problem is unbounded; a feasible solution has been computed.

OPTIMIZED The MIP problem is optimized; an optimal solution has been computed.

PARTIALLY_SATISFIABLE The feasible region of the MIP problem has been changed by adding new space dimensions or new constraints; a feasible solution for the old feasible region is still available.

Definition at line 454 of file MIP_Problem.defs.hh.

11.139.4 Constructor & Destructor Documentation

11.139.4.1 Parma_Polyhedra_Library::MIP_Problem::MIP_Problem (dimension_type *dim* = 0) [explicit]

Builds a trivial MIP problem.

A trivial MIP problem requires to maximize the objective function 0 on a vector space under no constraints at all: the origin of the vector space is an optimal solution.

Parameters

dim The dimension of the vector space enclosing `*this` (optional argument with default value 0).

Exceptions

std::length_error Thrown if `dim` exceeds `max_space_dimension()`.

Definition at line 65 of file MIP_Problem.cc.

References max_space_dimension(), and OK().

11.139.4.2 template<typename In > Parma_Polyhedra_Library::MIP_Problem::MIP_Problem (dimension_type *dim*, In *first*, In *last*, const Variables_Set & *int_vars*, const Linear_Expression & *obj* = `Linear_Expression::zero()`, Optimization_Mode *mode* = `MAXIMIZATION`) [inline]

Builds an MIP problem having space dimension `dim` from the sequence of constraints in the range $[\mathrm{first}, \mathrm{last})$, the objective function `obj` and optimization mode `mode`; those dimensions whose indices occur in `int_vars` are constrained to take an integer value.

Parameters

dim The dimension of the vector space enclosing `*this`.

first An input iterator to the start of the sequence of constraints.

last A past-the-end input iterator to the sequence of constraints.

int_vars The set of variables' indexes that are constrained to take integer values.

obj The objective function (optional argument with default value 0).

mode The optimization mode (optional argument with default value `MAXIMIZATION`).

Exceptions

std::length_error Thrown if `dim` exceeds `max_space_dimension()`.

std::invalid_argument Thrown if a constraint in the sequence is a strict inequality, if the space dimension of a constraint (resp., of the objective function or of the integer variables) or the space dimension of the integer variable set is strictly greater than `dim`.

Definition at line 32 of file MIP_Problem.templates.hh.

References external_space_dim, i_variables, input_cs, max_space_dimension(), OK(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and Parma_Polyhedra_Library::Variables_Set::space_dimension().

11.139.4.3 template<typename In > Parma_Polyhedra_Library::MIP_Problem::MIP_Problem (dimension_type *dim*, In *first*, In *last*, const Linear_Expression & *obj* = Linear_Expression::zero(), Optimization_Mode *mode* = MAXIMIZATION) [inline]

Builds an MIP problem having space dimension dim from the sequence of constraints in the range [first, last), the objective function obj and optimization mode mode.

Parameters

dim The dimension of the vector space enclosing *this.

first An input iterator to the start of the sequence of constraints.

last A past-the-end input iterator to the sequence of constraints.

obj The objective function (optional argument with default value 0).

mode The optimization mode (optional argument with default value MAXIMIZATION).

Exceptions

std::length_error Thrown if dim exceeds max_space_dimension().

std::invalid_argument Thrown if a constraint in the sequence is a strict inequality or if the space dimension of a constraint (resp., of the objective function or of the integer variables) is strictly greater than dim.

Definition at line 102 of file MIP_Problem.templates.hh.

References input_cs, max_space_dimension(), OK(), and Parma_Polyhedra_Library::Linear_-Expression::space_dimension().

11.139.4.4 Parma_Polyhedra_Library::MIP_Problem::MIP_Problem (dimension_type *dim*, const Constraint_System & *cs*, const Linear_Expression & *obj* = Linear_Expression::zero(), Optimization_Mode *mode* = MAXIMIZATION)

Builds an MIP problem having space dimension dim from the constraint system cs, the objective function obj and optimization mode mode.

Parameters

dim The dimension of the vector space enclosing *this.

cs The constraint system defining the feasible region.

obj The objective function (optional argument with default value 0).

mode The optimization mode (optional argument with default value MAXIMIZATION).

Exceptions

std::length_error Thrown if dim exceeds max_space_dimension().

std::invalid_argument Thrown if the constraint system contains any strict inequality or if the space dimension of the constraint system (resp., the objective function) is strictly greater than dim.

Definition at line 90 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), input_cs, max_space_dimension(), OK(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.139.4.5 Parma_Polyhedra_Library::MIP_Problem::MIP_Problem (const MIP_Problem & *y*) [inline]

Ordinary copy constructor.

Definition at line 44 of file MIP_Problem.inlines.hh.

References OK().

11.139.4.6 Parma_Polyhedra_Library::MIP_Problem::~MIP_Problem () [inline]

Destructor.

Definition at line 64 of file MIP_Problem.inlines.hh.

11.139.5 Member Function Documentation

11.139.5.1 void Parma_Polyhedra_Library::MIP_Problem::add_constraint (const Constraint & *c*)

Adds a copy of constraint `c` to the MIP problem.

Exceptions

std::invalid_argument Thrown if the constraint `c` is a strict inequality or if its space dimension is strictly greater than the space dimension of `*this`.

Definition at line 142 of file MIP_Problem.cc.

References input_cs, Parma_Polyhedra_Library::Constraint::is_strict_inequality(), OK(), PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::Constraint::space_dimension(), space_dimension(), status, and UNSATISFIABLE.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Box< ITV >::Box(), is_mip_satisfiable(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and solve_mip().

11.139.5.2 void Parma_Polyhedra_Library::MIP_Problem::add_constraints (const Constraint_System & *cs*)

Adds a copy of the constraints in `cs` to the MIP problem.

Exceptions

std::invalid_argument Thrown if the constraint system `cs` contains any strict inequality or if its space dimension is strictly greater than the space dimension of `*this`.

Definition at line 160 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), input_cs, OK(), PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::Constraint_System::space_dimension(), space_dimension(), status, and UNSATISFIABLE.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.139.5.3 void Parma_Polyhedra_Library::MIP_Problem::add_space_dimensions_and_embed (dimension_type *m*)

Adds `m` new space dimensions and embeds the old MIP problem in the new vector space.

Parameters

m The number of dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

The new space dimensions will be those having the highest indexes in the new MIP problem; they are initially unconstrained.

Definition at line 335 of file MIP_Problem.cc.

References external_space_dim, max_space_dimension(), OK(), PARTIALLY_SATISFIABLE, space_dimension(), status, and UNSATISFIABLE.

Referenced by Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.139.5.4 void Parma_Polyhedra_Library::MIP_Problem::add_to_integer_space_dimensions (const Variables_Set & *i_vars*)

Sets the variables whose indexes are in set `i_vars` to be integer space dimensions.

Exceptions

std::invalid_argument Thrown if some index in `i_vars` does not correspond to a space dimension in `*this`.

Definition at line 351 of file MIP_Problem.cc.

References external_space_dim, i_variables, Parma_Polyhedra_Library::Variables_Set::insert(), PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::Variables_Set::space_dimension(), status, and UNSATISFIABLE.

11.139.5.5 void Parma_Polyhedra_Library::MIP_Problem::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 2027 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Variables_Set::ascii_dump(), Parma_Polyhedra_Library::Generator::ascii_dump(), Parma_Polyhedra_Library::Row::ascii_dump(), Parma_Polyhedra_Library::Matrix::ascii_dump(), Parma_Polyhedra_Library::Linear_Expression::ascii_dump(), ascii_dump(), base, external_space_dim, first_pending_constraint, i_variables, initialized, input_cs, input_obj_function, internal_space_dim, last_generator, mapping, Parma_Polyhedra_Library::MAXIMIZATION, opt_mode, OPTIMIZED, PARTIALLY_SATISFIABLE, pricing, PRICING_STEEPEST_EDGE_EXACT, PRICING_STEEPEST_EDGE_FLOAT, PRICING_TEXTBOOK, SATISFIABLE, Parma_Polyhedra_Library::Row::size(), status, tableau, UNBOUNDED, UNSATISFIABLE, and working_cost.

11.139.5.6 void Parma_Polyhedra_Library::MIP_Problem::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by ascii_dump(), and OK().

11.139.5.7 bool Parma_Polyhedra_Library::MIP_Problem::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 2107 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Constraint::ascii_load(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, and Parma_Polyhedra_Library::Constraint::zero_dim_positivity().

11.139.5.8 bool Parma_Polyhedra_Library::MIP_Problem::choose_branching_variable (const MIP_Problem & *lp*, const Variables_Set & *i_vars*, dimension_type & *branching_index*) [static, private]

Returns `true` if and if only `mip.last_generator` satisfies all the integrality coditions implicitly stated using by `i_vars`.

Parameters

lp The LP problem. This is assumed to have no integral space dimension.

i_vars The variables that are constrained to take an integer value.

branching_index If `false` is returned, this will encode the non-integral variable index on which the 'branch and bound' algorithm should be applied.

Definition at line 1722 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::gcd_assign(), input_cs, Parma_Polyhedra_Library::Variables_Set::insert(), is_saturated(), last_generator, PPL_DIRTY_TEMP_COEFFICIENT, and space_dimension().

Referenced by is_mip_satisfiable().

11.139.5.9 void Parma_Polyhedra_Library::MIP_Problem::clear () [inline]

Resets `*this` to be equal to the trivial MIP problem.

The space dimension is reset to 0.

Definition at line 148 of file MIP_Problem.inlines.hh.

References swap().

11.139.5.10 void Parma_Polyhedra_Library::MIP_Problem::compute_generator () const [private]

Computes a valid generator that satisfies all the constraints of the Linear Programming problem associated to `*this`.

Definition at line 1386 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::exact_div_assign(), external_space_dim, is_in_base(), last_generator, Parma_Polyhedra_Library::lcm_assign(), mapping, Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::sub_mul_assign(), and tableau.

Referenced by process_pending_constraints(), and second_phase().

11.139.5.11 bool Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_exact_pricing () [private]

Returns `true` if and if only the algorithm successfully computed a feasible solution.

Note

Uses an exact pricing method (either textbook or exact steepest edge), so that the result is deterministic across different architectures.

Definition at line 1281 of file MIP_Problem.cc.

References get_control_parameter(), get_exiting_base_index(), Parma_Polyhedra_Library::maybe_abandon(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), pivot(), PRICING, PRICING_STEEPEST_EDGE_EXACT, PRICING_-

TEXTBOOK, Parma_Polyhedra_Library::Row::size(), steepest_edge_exact_entering_index(), tableau, textbook_entering_index(), and working_cost.

Referenced by process_pending_constraints(), and second_phase().

11.139.5.12 bool Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float () [private]

Returns `true` if and if only the algorithm successfully computed a feasible solution.

Note

Uses a floating point implementation of the steepest edge pricing method, so that the result is correct, but not deterministic across different architectures.

Definition at line 1190 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::abs_assign(), get_exiting_base_index(), Parma_Polyhedra_Library::maybe_abandon(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), pivot(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Row::size(), steepest_edge_float_entering_index(), tableau, textbook_entering_index(), WEIGHT_ADD, WEIGHT_BEGIN, and working_cost.

Referenced by process_pending_constraints(), and second_phase().

11.139.5.13 MIP_Problem_Status Parma_Polyhedra_Library::MIP_Problem::compute_tableau (std::vector< dimension_type > & *worked_out_row*) [private]

Assigns to `this->tableau` a simplex tableau representing the MIP problem, inserting into `this->mapping` the information that is required to recover the original MIP problem.

Returns

`UNFEASIBLE_MIP_PROBLEM` if the constraint system contains any trivially unfeasible constraint (tableau was not computed); `UNBOUNDED_MIP_PROBLEM` if the problem is trivially unbounded (the computed tableau contains no constraints); `OPTIMIZED_MIP_PROBLEM`> if the problem is neither trivially unfeasible nor trivially unbounded (the tableau was computed successfully).

11.139.5.14 MIP_Problem::const_iterator Parma_Polyhedra_Library::MIP_Problem::constraints_begin () const [inline]

Returns a read-only iterator to the first constraint defining the feasible region.

Definition at line 95 of file MIP_Problem.inlines.hh.

References input_cs.

Referenced by operator<<().

11.139.5.15 MIP_Problem::const_iterator Parma_Polyhedra_Library::MIP_Problem::constraints_end () const [inline]

Returns a past-the-end read-only iterator to the sequence of constraints defining the feasible region.

Definition at line 100 of file MIP_Problem.inlines.hh.

References input_cs.

Referenced by operator<<().

11.139.5.16 void Parma_Polyhedra_Library::MIP_Problem::erase_artificials (dimension_type *begin_artificials*, dimension_type *end_artificials*) [private]

Drop unnecessary artificial variables from the tableau and get ready for the second phase of the simplex algorithm.

Note

The two parameters denote a STL-like half-open range. It is assumed that `begin_artificials` is strictly greater than 0 and smaller than `end_artificials`.

Parameters

begin_artificials The start of the tableau column index range for artificial variables.

end_artificials The end of the tableau column index range for artificial variables. Note that column index end_artificial is *excluded* from the range.

Definition at line 1328 of file MIP_Problem.cc.

References base, Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), pivot(), Parma_Polyhedra_Library::Matrix::remove_trailing_columns(), Parma_Polyhedra_Library::Row::shrink(), Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Row::swap(), tableau, and working_cost.

Referenced by process_pending_constraints().

11.139.5.17 void Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function (const Generator & *evaluating_point*, Coefficient & *num*, Coefficient & *den*) const

Sets `num` and `den` so that $\frac{num}{den}$ is the result of evaluating the objective function on `evaluating_point`.

Parameters

evaluating_point The point on which the objective function will be evaluated.

num On exit will contain the numerator of the evaluated value.

den On exit will contain the denominator of the evaluated value.

Exceptions

std::invalid_argument Thrown if `*this` and `evaluating_point` are dimension-incompatible or if the generator `evaluating_point` is not a point.

Definition at line 1532 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), input_obj_function, Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::normalize2(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), space_dimension(), and Parma_Polyhedra_Library::Generator::space_dimension().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), optimal_value(), and solve_mip().

11.139.5.18 memory_size_type Parma_Polyhedra_Library::MIP_Problem::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 154 of file MIP_Problem.inlines.hh.

References base, Parma_Polyhedra_Library::Generator::external_memory_in_bytes(), Parma_Polyhedra_Library::Linear_Expression::external_memory_in_bytes(), Parma_Polyhedra_Library::Row::external_memory_in_bytes(), Parma_Polyhedra_Library::Matrix::external_memory_in_bytes(), input_cs, input_obj_function, last_generator, mapping, tableau, and working_cost.

Referenced by total_memory_in_bytes().

11.139.5.19 const PPL::Generator & Parma_Polyhedra_Library::MIP_Problem::feasible_point () const

Returns a feasible point for `*this`, if it exists.

Exceptions

std::domain_error Thrown if the MIP problem is not satisfiable.

Definition at line 195 of file MIP_Problem.cc.

References is_satisfiable(), and last_generator.

11.139.5.20 MIP_Problem::Control_Parameter_Value Parma_Polyhedra_Library::MIP_Problem::get_control_parameter (Control_Parameter_Name *name*) const [inline]

Returns the value of the control parameter `name`.

Definition at line 110 of file MIP_Problem.inlines.hh.

References pricing, and PRICING.

Referenced by compute_simplex_using_exact_pricing(), process_pending_constraints(), and second_phase().

11.139.5.21 PPL::dimension_type Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index (dimension_type *entering_var_index*) const [private]

Computes the row index of the variable exiting the base of the MIP problem. Implemented with anti-cycling rule.

Returns

The row index of the variable exiting the base.

Parameters

entering_var_index The column index of the variable entering the base.

Definition at line 1135 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::abs_assign(), base, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::lcm_assign(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, tableau, WEIGHT_ADD, and WEIGHT_BEGIN.

Referenced by compute_simplex_using_exact_pricing(), and compute_simplex_using_steepest_edge_float().

11.139.5.22 const Variables_Set & Parma_Polyhedra_Library::MIP_Problem::integer_space_dimensions () const [inline]

Returns a set containing all the variables' indexes constrained to be integral.

Definition at line 105 of file MIP_Problem.inlines.hh.

References i_variables.

Referenced by is_mip_satisfiable(), operator<<(), and solve().

11.139.5.23 bool Parma_Polyhedra_Library::MIP_Problem::is_in_base (dimension_type *var_index*, dimension_type & *row_index*) const [private]

Definition at line 366 of file MIP_Problem.cc.

References base.

Referenced by compute_generator(), and merge_split_variable().

11.139.5.24 bool Parma_Polyhedra_Library::MIP_Problem::is_lp_satisfiable () const [private]

Returns `true` if and if only the LP problem is satisfiable.

Definition at line 1560 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), external_space_dim, first_pending_-constraint, initialized, input_cs, internal_space_dim, mapping, Parma_Polyhedra_Library::Matrix::num_-columns(), OK(), OPTIMIZED, PARTIALLY_SATISFIABLE, process_pending_constraints(), SATISFI-ABLE, status, tableau, UNBOUNDED, and UNSATISFIABLE.

Referenced by is_mip_satisfiable(), is_satisfiable(), solve(), and solve_mip().

11.139.5.25 bool Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable (MIP_Problem & *lp*, const Variables_Set & *i_vars*, Generator & *p*) [static, private]

Returns `true` if and if only the LP problem `lp` is satisfiable when variables in `i_vars` can only take integral values.

Parameters

lp The LP problem. This is assumed to have no integral space dimension.

i_vars The variables that are constrained to take integral values.

p If `true` is returned, it will encode a feasible point.

Definition at line 1776 of file MIP_Problem.cc.

References add_constraint(), Parma_Polyhedra_Library::assign_r(), choose_branching_variable(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::gcd_assign(), integer_space_dimensions(), is_lp_satisfiable(), last_generator, PPL_DIRTY_TEMP_COEFFICIENT, and space_dimension().

Referenced by is_satisfiable().

11.139.5.26 bool Parma_Polyhedra_Library::MIP_Problem::is_satisfiable () const

Checks satisfiability of `*this`.

Returns

`true` if and only if the MIP problem is satisfiable.

Definition at line 213 of file MIP_Problem.cc.

References i_variables, Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_-Relaxation::i_vars, is_lp_satisfiable(), is_mip_satisfiable(), last_generator, Parma_Polyhedra_-Library::MIP_Problem::RAII_Temporary_Real_Relaxation::lp, OK(), OPTIMIZED, PARTIALLY_-SATISFIABLE, SATISFIABLE, status, UNBOUNDED, and UNSATISFIABLE.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::Box< ITV >::Box(), feasible_point(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.139.5.27 bool Parma_Polyhedra_Library::MIP_Problem::is_satisfied (const Constraint & *c*, const Generator & *g*) [static, private]

Returns `true` if and only if `c` is satisfied by `g`.

Definition at line 432 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Generator::space_dimension().

Referenced by OK(), and parse_constraints().

11.139.5.28 bool Parma_Polyhedra_Library::MIP_Problem::is_saturated (const Constraint & *c*, const Generator & *g*) [static, private]

Returns `true` if and only if `c` is saturated by `g`.

Definition at line 442 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Generator::space_dimension().

Referenced by choose_branching_variable().

11.139.5.29 void Parma_Polyhedra_Library::MIP_Problem::linear_combine (Row & *x*, const Row & *y*, const dimension_type *k*) [static, private]

Linearly combines `x` with `y` so that `*this[k]` is 0.

Parameters

- ***x*** The Row that will be combined with `y` object.
- ***y*** The Row that will be combined with `x` object.
- ***k*** The position of `*this` that have to be 0.

Computes a linear combination of `x` and `y` having the element of index `k` equal to 0. Then it assigns the resulting Linear_Row to `x` and normalizes it.

Definition at line 1086 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::normalize2(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::sub_mul_assign(), WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by pivot(), process_pending_constraints(), and second_phase().

11.139.5.30 dimension_type Parma_Polyhedra_Library::MIP_Problem::max_space_dimension () [inline, static]

Returns the maximum space dimension an MIP_Problem can handle.

Definition at line 33 of file MIP_Problem.inlines.hh.

Referenced by add_space_dimensions_and_embed(), and MIP_Problem().

11.139.5.31 PPL::dimension_type Parma_Polyhedra_Library::MIP_Problem::merge_split_variable (dimension_type *var_index*) [private]

Merges back the positive and negative part of a (previously split) variable after detecting a corresponding nonnegativity constraint.

Returns

If the negative part of `var_index` was in base, the index of the corresponding tableau row (which has become nonfeasible); otherwise `not_a_dimension()`.

Parameters

var_index The index of the variable that has to be merged.

Definition at line 375 of file MIP_Problem.cc.

References base, is_in_base(), mapping, Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::permute_columns(), Parma_Polyhedra_Library::Matrix::remove_trailing_columns(), and tableau.

Referenced by process_pending_constraints().

11.139.5.32 const Linear_Expression & Parma_Polyhedra_Library::MIP_Problem::objective_function () const [inline]

Returns the objective function.

Definition at line 79 of file MIP_Problem.inlines.hh.

References input_obj_function.

Referenced by operator<<().

11.139.5.33 bool Parma_Polyhedra_Library::MIP_Problem::OK () const

Checks if all the invariants are satisfied.

Definition at line 1872 of file MIP_Problem.cc.

References ascii_dump(), base, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), external_space_dim, first_pending_constraint, Parma_Polyhedra_Library::gcd_assign(), i_variables, initialized, input_cs, input_obj_function, is_satisfied(), last_generator, mapping, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator::OK(), Parma_Polyhedra_Library::Linear_Expression::OK(), Parma_Polyhedra_Library::Matrix::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), status, tableau, UNSATISFIABLE, and working_cost.

Referenced by add_constraint(), add_constraints(), add_space_dimensions_and_embed(), is_lp_satisfiable(), is_satisfiable(), MIP_Problem(), process_pending_constraints(), second_phase(), set_objective_function(), set_optimization_mode(), and solve().

11.139.5.34 MIP_Problem & Parma_Polyhedra_Library::MIP_Problem::operator= (const MIP_Problem & *y*) [inline]

Assignment operator.

Definition at line 141 of file MIP_Problem.inlines.hh.

References swap().

11.139.5.35 void Parma_Polyhedra_Library::MIP_Problem::optimal_value (Coefficient & *num*, Coefficient & *den*) const [inline]

Sets `num` and `den` so that $\frac{num}{den}$ is the solution of the optimization problem.

Exceptions

std::domain_error Thrown if `*this` doesn't not have an optimizing point, i.e., if the MIP problem is unbounded or not satisfiable.

Definition at line 89 of file MIP_Problem.inlines.hh.

References evaluate_objective_function(), and optimizing_point().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), and Parma_Polyhedra_Library::BD_Shape< T >::max_min().

11.139.5.36 Optimization_Mode Parma_Polyhedra_Library::MIP_Problem::optimization_mode () const [inline]

Returns the optimization mode.

Definition at line 84 of file MIP_Problem.inlines.hh.

References opt_mode.

Referenced by operator<<(), and solve_mip().

11.139.5.37 const PPL::Generator & Parma_Polyhedra_Library::MIP_Problem::optimizing_point () const

Returns an optimal point for `*this`, if it exists.

Exceptions

std::domain_error Thrown if `*this` doesn't not have an optimizing point, i.e., if the MIP problem is unbounded or not satisfiable.

Definition at line 204 of file MIP_Problem.cc.

References last_generator, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, and solve().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and optimal_value().

11.139.5.38 bool Parma_Polyhedra_Library::MIP_Problem::parse_constraints (dimension_type & *additional_tableau_rows*, dimension_type & *additional_slack_variables*, std::deque< bool > & *is_tableau_constraint*, std::deque< bool > & *is_satisfied_inequality*, std::deque< bool > & *is_nonnegative_variable*, std::deque< bool > & *is_remergeable_variable*) const [private]

Parses the pending constraints to gather information on how to resize the tableau.

Note

All of the method parameters are output parameters; their value is only meaningful when the function exit returning value `true`.

Returns

`false` if a trivially false constraint is detected, `true` otherwise.

Parameters

additional_tableau_rows On exit, this will store the number of rows that have to be added to the original tableau.

additional_slack_variables This will store the number of slack variables that have to be added to the original tableau.

is_tableau_constraint This container of Boolean flags is initially empty. On exit, it size will be equal to the number of pending constraints in `input_cs`. For each pending constraint index `i`, the corresponding element of this container (having index `i - first_pending_constraint`) will be set to `true` if and only if the constraint has to be included in the tableau.

is_satisfied_inequality This container of Boolean flags is initially empty. On exit, its size will be equal to the number of pending constraints in `input_cs`. For each pending constraint index `i`, the corresponding element of this container (having index `i - first_pending_constraint`) will be set to `true` if and only if it is an inequality and it is already satisfied by `last_generator` (hence it does not require the introduction of an artificial variable).

is_nonnegative_variable This container of Boolean flags is initially empty. On exit, it size is equal to `external_space_dim`. For each variable (index), the corresponding element of this container is `true` if the variable is known to be nonnegative (and hence should not be split into a positive and a negative part).

is_remergeable_variable This container of Boolean flags is initially empty. On exit, it size is equal to `internal_space_dim`. For each variable (index), the corresponding element of this container is `true` if the variable was previously split into positive and negative parts that can now be merged back, since it is known that the variable is nonnegative.

Definition at line 453 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Constraint::coefficient(), external_space_dim, first_pending_constraint, Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), input_cs, internal_space_-

dim, Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_inequality(), is_satisfied(), last_generator, mapping, and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by process_pending_constraints().

11.139.5.39 void Parma_Polyhedra_Library::MIP_Problem::pivot (dimension_type *entering_var_index*, dimension_type *exiting_base_index*) [private]

Performs the pivoting operation on the tableau.

Parameters

entering_var_index The index of the variable entering the base.

exiting_base_index The index of the row exiting the base.

Definition at line 1116 of file MIP_Problem.cc.

References base, linear_combine(), Parma_Polyhedra_Library::Matrix::num_rows(), tableau, and working_cost.

Referenced by compute_simplex_using_exact_pricing(), compute_simplex_using_steepest_edge_float(), and erase_artificials().

11.139.5.40 void Parma_Polyhedra_Library::MIP_Problem::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.139.5.41 bool Parma_Polyhedra_Library::MIP_Problem::process_pending_constraints () [private]

Processes the pending constraints of `*this`.

Returns

`true` if and only if the MIP problem is satisfiable after processing the pending constraints, `false` otherwise.

Definition at line 638 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::add_zero_rows(), base, Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Constraint::coefficient(), compute_generator(), compute_simplex_using_exact_pricing(), compute_simplex_using_steepest_edge_float(), erase_artificials(), external_space_dim, first_pending_constraint, get_control_parameter(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), input_cs, input_obj_function, internal_space_dim, Parma_Polyhedra_Library::Constraint::is_inequality(), last_generator, linear_combine(),

mapping, Parma_Polyhedra_Library::MAXIMIZATION, merge_split_variable(), Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), opt_mode, OPTIMIZED, parse_constraints(), PRICING, PRICING_STEEPEST_EDGE_FLOAT, SATISFIABLE, Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), status, tableau, UNBOUNDED, UNSATISFIABLE, and working_cost.

Referenced by is_lp_satisfiable().

11.139.5.42 void Parma_Polyhedra_Library::MIP_Problem::second_phase () [private]

Optimizes the MIP problem using the second phase of the primal simplex algorithm.

Definition at line 1474 of file MIP_Problem.cc.

References base, Parma_Polyhedra_Library::Linear_Expression::coefficient(), compute_generator(), compute_simplex_using_exact_pricing(), compute_simplex_using_steepest_edge_float(), get_control_parameter(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), input_obj_function, linear_combine(), mapping, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), opt_mode, OPTIMIZED, PRICING, PRICING_STEEPEST_EDGE_FLOAT, SATISFIABLE, Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), status, Parma_Polyhedra_Library::Row::swap(), tableau, UNBOUNDED, and working_cost.

Referenced by solve(), and solve_mip().

11.139.5.43 void Parma_Polyhedra_Library::MIP_Problem::set_control_parameter (Control_Parameter_Value *value*) [inline]

Sets control parameter `value`.

Definition at line 117 of file MIP_Problem.inlines.hh.

References pricing.

11.139.5.44 void Parma_Polyhedra_Library::MIP_Problem::set_objective_function (const Linear_Expression & *obj*)

Sets the objective function to `obj`.

Exceptions

std::invalid_argument Thrown if the space dimension of `obj` is strictly greater than the space dimension of `*this`.

Definition at line 179 of file MIP_Problem.cc.

References input_obj_function, OK(), OPTIMIZED, SATISFIABLE, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), space_dimension(), status, and UNBOUNDED.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.139.5.45 void Parma_Polyhedra_Library::MIP_Problem::set_optimization_mode (Optimization_Mode *mode*) [inline]

Sets the optimization mode to `mode`.

Definition at line 69 of file MIP_Problem.inlines.hh.

References OK(), opt_mode, OPTIMIZED, SATISFIABLE, status, and UNBOUNDED.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints().

11.139.5.46 PPL::MIP_Problem_Status Parma_Polyhedra_Library::MIP_Problem::solve () const

Optimizes the MIP problem.

Returns

An MIP_Problem_Status flag indicating the outcome of the optimization attempt (unfeasible, unbounded or optimized problem).

Definition at line 257 of file MIP_Problem.cc.

References i_variables, Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_-Relaxation::i_vars, integer_space_dimensions(), is_lp_satisfiable(), last_generator, Parma_Polyhedra_-Library::MIP_Problem::RAII_Temporary_Real_Relaxation::lp, OK(), OPTIMIZED, Parma_Polyhedra_-Library::OPTIMIZED_MIP_PROBLEM, PARTIALLY_SATISFIABLE, SATISFIABLE, second_phase(), solve_mip(), status, UNBOUNDED, Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM, and UNSATISFIABLE.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), optimizing_point(), Parma_-Polyhedra_Library::Polyhedron::simplify_using_context_assign(), and Parma_Polyhedra_-Library::Polyhedron::strongly_minimize_constraints().

11.139.5.47 PPL::MIP_Problem_Status Parma_Polyhedra_Library::MIP_Problem::solve_mip (bool & *have_incumbent_solution*, mpq_class & *incumbent_solution_value*, Generator & *incumbent_solution_point*, MIP_Problem & *mip*, const Variables_Set & *i_vars*) [static, private]

Returns a status that encodes the solution of the MIP problem.

Parameters

have_incumbent_solution It is used to store if the solving process has found a provisional optimum point.

incumbent_solution_value Encodes the evaluated value of the provisional optimum point found.

incumbent_solution_point If the method returns 'OPTIMIZED', this will contain the optimality point.

mip The problem that has to be solved.

i_vars The variables that are constrained to take an integer value.

Definition at line 1605 of file MIP_Problem.cc.

References add_constraint(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Generator::divisor(), evaluate_objective_function(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::is_canonical(), is_lp_satisfiable(), last_generator, Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, optimization_mode(), OPTIMIZED, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, second_phase(), space_dimension(), status, Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, and Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM.

Referenced by solve().

11.139.5.48 dimension_type Parma_Polyhedra_Library::MIP_Problem::space_dimension () const [inline]

Returns the space dimension of the MIP problem.

Definition at line 38 of file MIP_Problem.inlines.hh.

References external_space_dim.

Referenced by add_constraint(), add_constraints(), add_space_dimensions_and_embed(), choose_branching_variable(), evaluate_objective_function(), is_mip_satisfiable(), process_pending_constraints(), set_objective_function(), and solve_mip().

11.139.5.49 PPL::dimension_type Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index () const [private]

Computes the column index of the variable entering the base, using an exact steepest-edge algorithm with anti-cycling rule.

Returns

The column index of the variable that enters the base. If no such variable exists, optimality was achieved and `0` is returned.

To compute the entering_index, the steepest edge algorithm chooses the index 'j' such that $\frac{d_j}{\|\Delta x^j\|}$ is the largest in absolute value, where

$$\|\Delta x^j\| = \left(1 + \sum_{i=1}^{m} \alpha_{ij}^2\right)^{\frac{1}{2}}.$$

Recall that, due to the exact integer implementation of the algorithm, our tableau doesn't contain the "real" α values, but these can be computed dividing the value of the coefficient by the value of the variable in base. Obviously the result may not be an integer, so we will proceed in another way: we compute the lcm of all the variables in base to get the good "weight" of each Coefficient of the tableau.

Definition at line 989 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::add_mul_assign(), base, Parma_Polyhedra_Library::exact_-div_assign(), Parma_Polyhedra_Library::lcm_assign(), Parma_Polyhedra_Library::Matrix::num_-columns(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Row::size(), swap(), tableau, WEIGHT_ADD_MUL, WEIGHT_BEGIN, and working_cost.

Referenced by compute_simplex_using_exact_pricing().

11.139.5.50 PPL::dimension_type Parma_Polyhedra_Library::MIP_Problem::steepest_edge_-float_entering_index () const [private]

Same as steepest_edge_exact_entering_index, but using floating points.

Note

Due to rounding errors, the index of the variable entering the base of the MIP problem is not predictable across different architectures. Hence, the overall simplex computation may differ in the path taken to reach the optimum. Anyway, the exact final result will be computed for the MIP_Problem.

Definition at line 943 of file MIP_Problem.cc.

References base, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_-Library::Matrix::num_rows(), Parma_Polyhedra_Library::Row::size(), tableau, WEIGHT_ADD_MUL, WEIGHT_BEGIN, and working_cost.

Referenced by compute_simplex_using_steepest_edge_float().

11.139.5.51 void Parma_Polyhedra_Library::MIP_Problem::swap (MIP_Problem & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 122 of file MIP_Problem.inlines.hh.

References base, external_space_dim, first_pending_constraint, i_variables, initialized, input_cs, input_-obj_function, internal_space_dim, last_generator, mapping, opt_mode, pricing, status, tableau, and working_cost.

Referenced by clear(), operator=(), Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_-Relaxation::RAII_Temporary_Real_Relaxation(), steepest_edge_exact_entering_index(), swap(), and Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation::~RAII_Temporary_-Real_Relaxation().

11.139.5.52 PPL::dimension_type Parma_Polyhedra_Library::MIP_Problem::textbook_-entering_index () const [private]

Computes the column index of the variable entering the base, using the textbook algorithm with anti-cycling rule.

Returns

The column index of the variable that enters the base. If no such variable exists, optimality was achieved and `0` is returned.

Definition at line 1067 of file MIP_Problem.cc.

References Parma_Polyhedra_Library::Row::size(), and working_cost.

Referenced by compute_simplex_using_exact_pricing(), and compute_simplex_using_steepest_edge_-float().

11.139.5.53 memory_size_type Parma_Polyhedra_Library::MIP_Problem::total_memory_in_-bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 173 of file MIP_Problem.inlines.hh.

References external_memory_in_bytes().

11.139.6 Friends And Related Function Documentation

11.139.6.1 std::ostream & operator<< (std::ostream & *s*, const MIP_Problem & *lp*) [related]

Output operator.

Definition at line 2296 of file MIP_Problem.cc.

References constraints_begin(), constraints_end(), integer_space_dimensions(), Parma_Polyhedra_-Library::MAXIMIZATION, objective_function(), and optimization_mode().

11.139.6.2 friend class RAII_Temporary_Real_Relaxation [friend]

Definition at line 525 of file MIP_Problem.defs.hh.

11.139.6.3 void swap (Parma_Polyhedra_Library::MIP_Problem & *x*, Parma_Polyhedra_Library::MIP_Problem & *y*) [related]

Specializes `std::swap`.

Definition at line 183 of file MIP_Problem.inlines.hh.

References swap().

11.139.7 Member Data Documentation

11.139.7.1 std::vector<dimension_type> Parma_Polyhedra_Library::MIP_Problem::base [private]

The current basic solution.

Definition at line 451 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), erase_artificials(), external_memory_in_bytes(), get_exiting_base_index(), is_in_base(), merge_split_variable(), OK(), pivot(), process_pending_constraints(), second_phase(), steepest_edge_exact_entering_index(), steepest_edge_float_entering_index(), and swap().

11.139.7.2 dimension_type Parma_Polyhedra_Library::MIP_Problem::external_space_dim [private]

The dimension of the vector space.

Definition at line 426 of file MIP_Problem.defs.hh.

Referenced by add_space_dimensions_and_embed(), add_to_integer_space_dimensions(), ascii_dump(), compute_generator(), is_lp_satisfiable(), MIP_Problem(), OK(), parse_constraints(), process_pending_-constraints(), space_dimension(), and swap().

11.139.7.3 dimension_type Parma_Polyhedra_Library::MIP_Problem::first_pending_constraint [private]

The first index of 'input_cs' containing a pending constraint.

Definition at line 492 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), is_lp_satisfiable(), OK(), parse_constraints(), process_pending_constraints(), and swap().

11.139.7.4 Variables_Set Parma_Polyhedra_Library::MIP_Problem::i_variables [private]

A set containing all the indexes of variables that are constrained to have an integer value.

Definition at line 507 of file MIP_Problem.defs.hh.

Referenced by add_to_integer_space_dimensions(), ascii_dump(), integer_space_dimensions(), is_-satisfiable(), MIP_Problem(), OK(), Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_-Relaxation::RAII_Temporary_Real_Relaxation(), solve(), swap(), and Parma_Polyhedra_Library::MIP_-Problem::RAII_Temporary_Real_Relaxation::~RAII_Temporary_Real_Relaxation().

11.139.7.5 bool Parma_Polyhedra_Library::MIP_Problem::initialized [private]

A Boolean encoding whether or not internal data structures have already been properly sized and populated: useful to allow for deeper checks in method OK().

Definition at line 486 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), is_lp_satisfiable(), OK(), and swap().

11.139.7.6 Constraint_Sequence Parma_Polyhedra_Library::MIP_Problem::input_cs [private]

The sequence of constraints describing the feasible region.

Definition at line 489 of file MIP_Problem.defs.hh.

Referenced by add_constraint(), add_constraints(), ascii_dump(), choose_branching_variable(), constraints_begin(), constraints_end(), external_memory_in_bytes(), is_lp_satisfiable(), MIP_Problem(), OK(), parse_constraints(), process_pending_constraints(), and swap().

11.139.7.7 Linear_Expression Parma_Polyhedra_Library::MIP_Problem::input_obj_function [private]

The objective function to be optimized.

Definition at line 495 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), evaluate_objective_function(), external_memory_in_bytes(), objective_function(), OK(), process_pending_constraints(), second_phase(), set_objective_function(), and swap().

11.139.7.8 dimension_type Parma_Polyhedra_Library::MIP_Problem::internal_space_dim [private]

The space dimension of the current (partial) solution of the MIP problem; it may be smaller than `external_space_dim`.

Definition at line 432 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), is_lp_satisfiable(), parse_constraints(), process_pending_constraints(), and swap().

11.139.7.9 Generator Parma_Polyhedra_Library::MIP_Problem::last_generator [private]

The last successfully computed feasible or optimizing point.

Definition at line 501 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), choose_branching_variable(), compute_generator(), external_memory_in_bytes(), feasible_point(), is_mip_satisfiable(), is_satisfiable(), OK(), optimizing_point(), parse_constraints(), process_pending_constraints(), solve(), solve_mip(), and swap().

11.139.7.10 std::vector<std::pair<dimension_type, dimension_type> > Parma_Polyhedra_Library::MIP_Problem::mapping [private]

A map between the variables of 'input_cs' and 'tableau'.

Contains all the pairs (i, j) such that mapping[i].first encodes the index of the column in the tableau where input_cs[i] is stored; mapping[i].second not a zero, encodes the split part of the tableau of input_cs[i]. The "positive" one is represented by mapping[i].first and the "negative" one is represented by mapping[i].second.

Definition at line 448 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), compute_generator(), external_memory_in_bytes(), is_lp_satisfiable(), merge_split_variable(), OK(), parse_constraints(), process_pending_constraints(), second_phase(), and swap().

11.139.7.11 Optimization_Mode Parma_Polyhedra_Library::MIP_Problem::opt_mode [private]

The optimization mode requested.

Definition at line 498 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), optimization_mode(), process_pending_constraints(), second_phase(), set_optimization_mode(), and swap().

11.139.7.12 Control_Parameter_Value Parma_Polyhedra_Library::MIP_Problem::pricing [private]

The pricing method in use.

Definition at line 479 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), get_control_parameter(), set_control_parameter(), and swap().

11.139.7.13 Status Parma_Polyhedra_Library::MIP_Problem::status [private]

The internal state of the MIP problem.

Definition at line 472 of file MIP_Problem.defs.hh.

Referenced by add_constraint(), add_constraints(), add_space_dimensions_and_embed(), add_to_integer_space_dimensions(), ascii_dump(), is_lp_satisfiable(), is_satisfiable(), OK(), process_pending_constraints(), second_phase(), set_objective_function(), set_optimization_mode(), solve(), solve_mip(), and swap().

11.139.7.14 Matrix Parma_Polyhedra_Library::MIP_Problem::tableau [private]

The matrix encoding the current feasible region in tableau form.

Definition at line 435 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), compute_generator(), compute_simplex_using_exact_pricing(), compute_simplex_using_steepest_edge_float(), erase_artificials(), external_memory_in_bytes(), get_exiting_base_index(), is_lp_satisfiable(), merge_split_variable(), OK(), pivot(), process_pending_constraints(), second_phase(), steepest_edge_exact_entering_index(), steepest_edge_float_entering_index(), and swap().

11.139.7.15 Row Parma_Polyhedra_Library::MIP_Problem::working_cost [private]

The working cost function.

Definition at line 438 of file MIP_Problem.defs.hh.

Referenced by ascii_dump(), compute_simplex_using_exact_pricing(), compute_simplex_using_steepest_edge_float(), erase_artificials(), external_memory_in_bytes(), OK(), pivot(), process_pending_constraints(), second_phase(), steepest_edge_exact_entering_index(), steepest_edge_float_entering_index(), swap(), and textbook_entering_index().

The documentation for this class was generated from the following files:

- MIP_Problem.defs.hh
- MIP_Problem.inlines.hh
- MIP_Problem.templates.hh
- MIP_Problem.cc

11.140 Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Public Types

- typedef P Policy

Static Public Member Functions

- static const T & raw_value (const Checked_Number< T, P > &v)

11.140.1 Detailed Description

template<typename T, typename P> struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >

Definition at line 102 of file Checked_Number.defs.hh.

11.140.2 Member Typedef Documentation

11.140.2.1 template<typename T , typename P > typedef P Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >::Policy

Definition at line 103 of file Checked_Number.defs.hh.

11.140.3 Member Function Documentation

11.140.3.1 template<typename T , typename P > static const T& Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >::raw_value (const Checked_Number< T, P > & *v*) [inline, static]

Definition at line 104 of file Checked_Number.defs.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::raw_value().

The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

11.141 Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Public Types

- typedef Checked_Number_Transparent_Policy< T > Policy

Static Public Member Functions

- static const T & raw_value (const T &v)

11.141.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >

Definition at line 91 of file Checked_Number.defs.hh.

11.141.2 Member Typedef Documentation

11.141.2.1 template<typename T > typedef Checked_Number_Transparent_Policy<T> Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >::Policy

Definition at line 92 of file Checked_Number.defs.hh.

11.141.3 Member Function Documentation

11.141.3.1 template<typename T > static const T& Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >::raw_value (const T & *v*) [inline, static]

Definition at line 93 of file Checked_Number.defs.hh.

References Parma_Polyhedra_Library::raw_value().

The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

11.142 Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Public Types

- typedef P Policy

Static Public Member Functions

- static T & raw_value (Checked_Number< T, P > &v)

11.142.1 Detailed Description

template<typename T, typename P> struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >

Definition at line 130 of file Checked_Number.defs.hh.

11.142.2 Member Typedef Documentation

11.142.2.1 template<typename T , typename P > typedef P Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >::Policy

Definition at line 131 of file Checked_Number.defs.hh.

11.142.3 Member Function Documentation

11.142.3.1 template<typename T , typename P > static T& Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >::raw_value (Checked_Number< T, P > & *v*) [inline, static]

Definition at line 132 of file Checked_Number.defs.hh.

References Parma_Polyhedra_Library::Checked_Number< T, Policy >::raw_value(), and Parma_Polyhedra_Library::raw_value().

The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

11.143 Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Public Types

- typedef Check_Overflow_Policy< T > Policy

Static Public Member Functions

- static T & raw_value (T &v)

11.143.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >

Definition at line 119 of file Checked_Number.defs.hh.

11.143.2 Member Typedef Documentation

11.143.2.1 template<typename T > typedef Check_Overflow_Policy<T> Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >::Policy

Definition at line 120 of file Checked_Number.defs.hh.

11.143.3 Member Function Documentation

11.143.3.1 template<typename T > static T& Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >::raw_value (T & *v*) [inline, static]

Definition at line 121 of file Checked_Number.defs.hh.

References Parma_Polyhedra_Library::raw_value().

The documentation for this struct was generated from the following file:

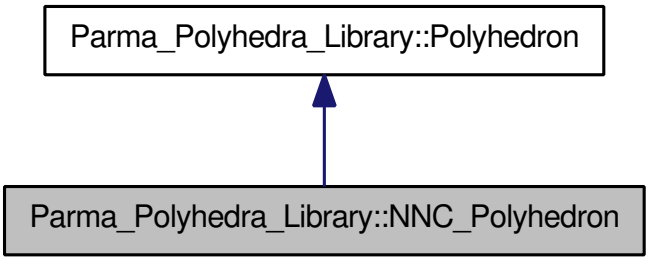
- Checked_Number.defs.hh

11.144 Parma_Polyhedra_Library::NNC_Polyhedron Class Reference

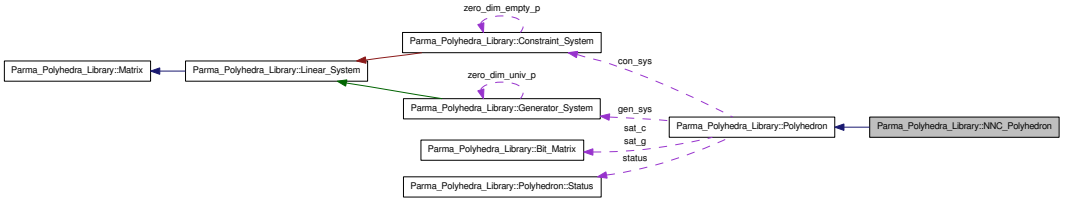
A not necessarily closed convex polyhedron.

`#include <NNC_Polyhedron.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::NNC_Polyhedron:



Collaboration diagram for Parma_Polyhedra_Library::NNC_Polyhedron:



Public Member Functions

- NNC_Polyhedron (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

  Builds either the universe or the empty NNC polyhedron.

- NNC_Polyhedron (const Constraint_System &cs)

  Builds an NNC polyhedron from a system of constraints.

- NNC_Polyhedron (Constraint_System &cs, Recycle_Input dummy)

  Builds an NNC polyhedron recycling a system of constraints.

- NNC_Polyhedron (const Generator_System &gs)

  Builds an NNC polyhedron from a system of generators.

- NNC_Polyhedron (Generator_System &gs, Recycle_Input dummy)

  Builds an NNC polyhedron recycling a system of generators.

- NNC_Polyhedron (const Congruence_System &cgs)

  Builds an NNC polyhedron from a system of congruences.

- NNC_Polyhedron (Congruence_System &cgs, Recycle_Input dummy)

  Builds an NNC polyhedron recycling a system of congruences.

- NNC_Polyhedron (const C_Polyhedron &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds an NNC polyhedron from the C polyhedron `y`.

- template<typename Interval >
  NNC_Polyhedron (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds an NNC polyhedron out of a box.

- NNC_Polyhedron (const Grid &grid, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds an NNC polyhedron out of a grid.

- template<typename U >
  NNC_Polyhedron (const BD_Shape< U > &bd, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a NNC polyhedron out of a BD shape.

- template<typename U >
  NNC_Polyhedron (const Octagonal_Shape< U > &os, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a NNC polyhedron out of an octagonal shape.

- NNC_Polyhedron (const NNC_Polyhedron &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- NNC_Polyhedron & operator= (const NNC_Polyhedron &y)

  The assignment operator. (`*this` *and* `y` *can be dimension-incompatible.).*

- NNC_Polyhedron & operator= (const C_Polyhedron &y)

  Assigns to `*this` *the C polyhedron* `y`*.*

- ~NNC_Polyhedron ()

  Destructor.

- bool poly_hull_assign_if_exact (const NNC_Polyhedron &y)

  If the poly-hull of `*this` *and* `y` *is exact it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- bool upper_bound_assign_if_exact (const NNC_Polyhedron &y)

  Same as poly_hull_assign_if_exact(y).

11.144.1 Detailed Description

A not necessarily closed convex polyhedron. An object of the class NNC_Polyhedron represents a *not necessarily closed* (NNC) convex polyhedron in the vector space $\mathbb{R}^n$.

Note

Since NNC polyhedra are a generalization of closed polyhedra, any object of the class C_Polyhedron can be (explicitly) converted into an object of the class NNC_Polyhedron. The reason for defining two different classes is that objects of the class C_Polyhedron are characterized by a more efficient implementation, requiring less time and memory resources.

Definition at line 46 of file NNC_Polyhedron.defs.hh.

11.144.2 Constructor & Destructor Documentation

11.144.2.1 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds either the universe or the empty NNC polyhedron.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the NNC polyhedron;

kind Specifies whether a universe or an empty NNC polyhedron should be built.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Both parameters are optional: by default, a 0-dimension space universe NNC polyhedron is built.

Definition at line 36 of file NNC_Polyhedron.inlines.hh.

11.144.2.2 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Constraint_System & *cs*) [inline, explicit]

Builds an NNC polyhedron from a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the polyhedron.

Definition at line 50 of file NNC_Polyhedron.inlines.hh.

11.144.2.3 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (Constraint_System & *cs*, Recycle_Input *dummy*) [inline]

Builds an NNC polyhedron recycling a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

cs The system of constraints defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Definition at line 62 of file NNC_Polyhedron.inlines.hh.

11.144.2.4 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Generator_System & *gs*) [inline, explicit]

Builds an NNC polyhedron from a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

gs The system of generators defining the polyhedron.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 75 of file NNC_Polyhedron.inlines.hh.

11.144.2.5 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (Generator_System & *gs*, Recycle_Input *dummy*) [inline]

Builds an NNC polyhedron recycling a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

gs The system of generators defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 87 of file NNC_Polyhedron.inlines.hh.

11.144.2.6 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Congruence_System & *cgs*) `[explicit]`

Builds an NNC polyhedron from a system of congruences.

The polyhedron inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

Definition at line 39 of file NNC_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_congruences(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.144.2.7 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (Congruence_System & *cgs*, Recycle_Input *dummy*)

Builds an NNC polyhedron recycling a system of congruences.

The polyhedron inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Definition at line 53 of file NNC_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_congruences(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.144.2.8 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const C_Polyhedron & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [explicit]

Builds an NNC polyhedron from the C polyhedron `y`.

Parameters

y The C polyhedron to be used;

complexity This argument is ignored.

Definition at line 33 of file NNC_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), Parma_Polyhedra_Library::Polyhedron::constraints(), and Parma_Polyhedra_Library::Polyhedron::OK().

11.144.2.9 template<typename Interval > Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds an NNC polyhedron out of a box.

The polyhedron inherits the space dimension of the box and is the most precise that includes the box.

Parameters

box The box representing the polyhedron to be built;

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 101 of file NNC_Polyhedron.inlines.hh.

11.144.2.10 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Grid & *grid*, Complexity_Class *complexity* = ANY_COMPLEXITY) [explicit]

Builds an NNC polyhedron out of a grid.

The polyhedron inherits the space dimension of the grid and is the most precise that includes the grid.

Parameters

grid The grid used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 68 of file NNC_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::Grid::constraints().

11.144.2.11 template<typename U > Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const BD_Shape< U > & *bd*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a NNC polyhedron out of a BD shape.

The polyhedron inherits the space dimension of the BD shape and is the most precise that includes the BD shape.

Parameters

bd The BD shape used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 114 of file NNC_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::BD_Shape< T >::constraints().

11.144.2.12 template<typename U > Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const Octagonal_Shape< U > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a NNC polyhedron out of an octagonal shape.

The polyhedron inherits the space dimension of the octagonal shape and is the most precise that includes the octagonal shape.

Parameters

os The octagonal shape used to build the polyhedron.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Definition at line 129 of file NNC_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraints(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints().

11.144.2.13 Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron (const NNC_Polyhedron & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

Definition at line 143 of file NNC_Polyhedron.inlines.hh.

11.144.2.14 Parma_Polyhedra_Library::NNC_Polyhedron::~NNC_Polyhedron () [inline]

Destructor.

Definition at line 32 of file NNC_Polyhedron.inlines.hh.

11.144.3 Member Function Documentation

11.144.3.1 NNC_Polyhedron & Parma_Polyhedra_Library::NNC_Polyhedron::operator= (const C_Polyhedron & *y*) [inline]

Assigns to `*this` the C polyhedron `y`.

Definition at line 154 of file NNC_Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::swap().

11.144.3.2 NNC_Polyhedron & Parma_Polyhedra_Library::NNC_Polyhedron::operator= (const NNC_Polyhedron & *y*) [inline]

The assignment operator. (`*this` and `y` can be dimension-incompatible.).

Definition at line 148 of file NNC_Polyhedron.inlines.hh.

11.144.3.3 bool Parma_Polyhedra_Library::NNC_Polyhedron::poly_hull_assign_if_exact (const NNC_Polyhedron & *y*)

If the poly-hull of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 82 of file NNC_Polyhedron.cc.

References Parma_Polyhedra_Library::Polyhedron::BHZ09_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::space_dimension(), and Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible().

Referenced by upper_bound_assign_if_exact().

11.144.3.4 bool Parma_Polyhedra_Library::NNC_Polyhedron::upper_bound_assign_if_exact (const NNC_Polyhedron & *y*) [inline]

Same as poly_hull_assign_if_exact(y).

Definition at line 161 of file NNC_Polyhedron.inlines.hh.

References poly_hull_assign_if_exact().

The documentation for this class was generated from the following files:

- NNC_Polyhedron.defs.hh
- NNC_Polyhedron.inlines.hh
- NNC_Polyhedron.cc

11.145 Parma_Polyhedra_Library::PIP_Solution_Node::No_Constraints Struct Reference

A tag type to select the alternative copy constructor.

`#include <PIP_Tree.defs.hh>`

11.145.1 Detailed Description

A tag type to select the alternative copy constructor.

Definition at line 603 of file PIP_Tree.defs.hh.

The documentation for this struct was generated from the following file:

- PIP_Tree.defs.hh

11.146 Parma_Polyhedra_Library::No_Reduction< D1, D2 > Class Template Reference

This class provides the reduction method for the Direct_Product domain.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- No_Reduction ()

  Default constructor.

- void product_reduce (D1 &d1, D2 &d2)

  The null reduction operator.

- ~No_Reduction ()

  Destructor.

11.146.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::No_Reduction< D1, D2 >

This class provides the reduction method for the Direct_Product domain. The reduction classes are used to instantiate the Partially_Reduced_Product domain template parameter `R`. This class does no reduction at all.

Definition at line 283 of file Partially_Reduced_Product.defs.hh.

11.146.2 Constructor & Destructor Documentation

11.146.2.1 template<typename D1 , typename D2 > Parma_Polyhedra_Library::No_Reduction< D1, D2 >::No_Reduction () [inline]

Default constructor.

Definition at line 747 of file Partially_Reduced_Product.inlines.hh.

11.146.2.2 template<typename D1 , typename D2 > Parma_Polyhedra_Library::No_Reduction< D1, D2 >::~No_Reduction () [inline]

Destructor.

Definition at line 756 of file Partially_Reduced_Product.inlines.hh.

11.146.3 Member Function Documentation

11.146.3.1 template<typename D1 , typename D2 > void Parma_Polyhedra_Library::No_Reduction< D1, D2 >::product_reduce (D1 & *d1*, D2 & *d2*) [inline]

The null reduction operator.

The parameters `d1` and `d2` are ignored.

Definition at line 751 of file Partially_Reduced_Product.inlines.hh.

The documentation for this class was generated from the following files:

- Partially_Reduced_Product.defs.hh
- Partially_Reduced_Product.inlines.hh

11.147 Parma_Polyhedra_Library::Not_A_Number Struct Reference

`#include <checked.defs.hh>`

Static Public Attributes

- static const Result_Class vclass = VC_NAN

11.147.1 Detailed Description

Definition at line 546 of file checked.defs.hh.

11.147.2 Member Data Documentation

11.147.2.1 const Result_Class Parma_Polyhedra_Library::Not_A_Number::vclass = VC_NAN [static]

Definition at line 547 of file checked.defs.hh.

The documentation for this struct was generated from the following file:

- checked.defs.hh

11.148 Parma_Polyhedra_Library::Checked::number_struct Struct Reference

Public Attributes

- unsigned int base
- bool neg_mantissa
- bool neg_exponent
- std::string mantissa
- unsigned int base_for_exponent
- unsigned long exponent

11.148.1 Detailed Description

Definition at line 39 of file checked.cc.

11.148.2 Member Data Documentation

11.148.2.1 unsigned int Parma_Polyhedra_Library::Checked::number_struct::base

Definition at line 40 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), Parma_Polyhedra_Library::Checked::parse_number(), and Parma_Polyhedra_Library::Checked::parse_number_part().

11.148.2.2 unsigned int Parma_Polyhedra_Library::Checked::number_struct::base_for_exponent

Definition at line 44 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), Parma_Polyhedra_Library::Checked::parse_number(), and Parma_Polyhedra_Library::Checked::parse_number_part().

11.148.2.3 unsigned long Parma_Polyhedra_Library::Checked::number_struct::exponent

Definition at line 45 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), Parma_Polyhedra_-Library::Checked::parse_number(), and Parma_Polyhedra_Library::Checked::parse_number_part().

11.148.2.4 std::string Parma_Polyhedra_Library::Checked::number_struct::mantissa

Definition at line 43 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), and Parma_Polyhedra_-Library::Checked::parse_number_part().

11.148.2.5 bool Parma_Polyhedra_Library::Checked::number_struct::neg_exponent

Definition at line 42 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), Parma_Polyhedra_-Library::Checked::parse_number(), and Parma_Polyhedra_Library::Checked::parse_number_part().

11.148.2.6 bool Parma_Polyhedra_Library::Checked::number_struct::neg_mantissa

Definition at line 41 of file checked.cc.

Referenced by Parma_Polyhedra_Library::Checked::input_mpq(), and Parma_Polyhedra_-Library::Checked::parse_number_part().

The documentation for this struct was generated from the following file:

- checked.cc

11.149 Parma_Polyhedra_Library::Octagonal_Shape< T > Class Template Reference

An octagonal shape.

`#include <Octagonal_Shape.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::Octagonal_Shape< T >:

Classes

- class Status

  A conjunctive assertion about a Octagonal_Shape<T> object.

Public Types

- typedef T coefficient_type_base

  The numeric base type upon which OSs are built.

- typedef N coefficient_type

  The (extended) numeric type of the inhomogeneous term of the inequalities defining an OS.

Public Member Functions

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- int32_t hash_code () const

  Returns a 32-bit hash code for `*this`.

Constructors, Assignment, Swap and Destructor

- Octagonal_Shape (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

  Builds an universe or empty OS of the specified space dimension.

- Octagonal_Shape (const Octagonal_Shape &x, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- template<typename U >
  Octagonal_Shape (const Octagonal_Shape< U > &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a conservative, upward approximation of `y`.

- Octagonal_Shape (const Constraint_System &cs)

  Builds an OS from the system of constraints `cs`.

- Octagonal_Shape (const Congruence_System &cgs)
  Builds an OS from a system of congruences.
- Octagonal_Shape (const Generator_System &gs)
  Builds an OS from the system of generators `gs`.
- Octagonal_Shape (const Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds an OS from the polyhedron `ph`.
- template<typename Interval >
  Octagonal_Shape (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds an OS out of a box.
- Octagonal_Shape (const Grid &grid, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds an OS that approximates a grid.
- template<typename U >
  Octagonal_Shape (const BD_Shape< U > &bd, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds an OS from a BD shape.
- Octagonal_Shape & operator= (const Octagonal_Shape &y)
  The assignment operator. (`*this` *and* `y` *can be dimension-incompatible.).*
- void swap (Octagonal_Shape &y)
  Swaps `*this` *with octagon* `y`*. (*`*this` *and* `y` *can be dimension-incompatible.).*
- ~Octagonal_Shape ()
  Destructor.

Member Functions that Do Not Modify the Octagonal_Shape

- dimension_type space_dimension () const
  Returns the dimension of the vector space enclosing `*this`.
- dimension_type affine_dimension () const
  Returns 0*, if* `*this` *is empty; otherwise, returns the affine dimension of* `*this`.
- Constraint_System constraints () const
  Returns the system of constraints defining `*this`.
- Constraint_System minimized_constraints () const
  Returns a minimized system of constraints defining `*this`.
- Congruence_System congruences () const
  Returns a system of (equality) congruences satisfied by `*this`.
- Congruence_System minimized_congruences () const
  Returns a minimal system of (equality) congruences satisfied by `*this` *with the same affine dimension as* `*this`.
- bool contains (const Octagonal_Shape &y) const

Returns `true` *if and only if* `*this` *contains* `y`.

- bool strictly_contains (const Octagonal_Shape &y) const
  Returns `true` *if and only if* `*this` *strictly contains* `y`.
- bool is_disjoint_from (const Octagonal_Shape &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*
- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between `*this` *and the constraint* `c`.
- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between `*this` *and the congruence* `cg`.
- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between `*this` *and the generator* `g`.
- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty OS.*
- bool is_universe () const
  Returns `true` *if and only if* `*this` *is a universe OS.*
- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*
- bool is_bounded () const
  Returns `true` *if and only if* `*this` *is a bounded OS.*
- bool is_topologically_closed () const
  Returns `true` *if and only if* `*this` *is a topologically closed subset of the vector space.*
- bool contains_integer_point () const
  Returns `true` *if and only if* `*this` *contains (at least) an integer point.*
- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.
- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from above in* `*this`.
- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from below in* `*this`.
- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*
- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*
- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const

Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &g) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool frequency (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const

  Returns `true` *if and only if there exist a unique value* `val` *such that* `*this` *saturates the equality* `expr = val`*.

- bool OK () const

  Checks if all the invariants are satisfied.

Space-Dimension Preserving Member Functions that May Modify the Octagonal_Shape

- void add_constraint (const Constraint &c)

  Adds a copy of constraint `c` *to the system of constraints defining* `*this`*.

- void add_constraints (const Constraint_System &cs)

  Adds the constraints in `cs` *to the system of constraints defining* `*this`*.

- void add_recycled_constraints (Constraint_System &cs)

  Adds the constraints in `cs` *to the system of constraints of* `*this`*.

- void add_congruence (const Congruence &cg)

  Adds to `*this` *a constraint equivalent to the congruence* `cg`*.

- void add_congruences (const Congruence_System &cgs)

  Adds to `*this` *constraints equivalent to the congruences in* `cgs`*.

- void add_recycled_congruences (Congruence_System &cgs)

  Adds to `*this` *constraints equivalent to the congruences in* `cgs`*.

- void refine_with_constraint (const Constraint &c)

  Uses a copy of constraint `c` *to refine the system of octagonal constraints defining* `*this`*.

- void refine_with_congruence (const Congruence &cg)

  Uses a copy of congruence `cg` *to refine the system of octagonal constraints of* `*this`*.

- void refine_with_constraints (const Constraint_System &cs)

  Uses a copy of the constraints in `cs` *to refine the system of octagonal constraints defining* `*this`*.

- void refine_with_congruences (const Congruence_System &cgs)

  Uses a copy of the congruences in `cgs` *to refine the system of octagonal constraints defining* `*this`*.

- void unconstrain (Variable var)

  Computes the cylindrification of `*this` *with respect to space dimension* `var`*, assigning the result to* `*this`*.

- void unconstrain (const Variables_Set &vars)

  Computes the cylindrification of `*this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `*this`*.

- void intersection_assign (const Octagonal_Shape &y)
  Assigns to `*this` *the intersection of* `*this` *and* `y`.

- void upper_bound_assign (const Octagonal_Shape &y)
  Assigns to `*this` *the smallest OS that contains the convex union of* `*this` *and* `y`.

- bool upper_bound_assign_if_exact (const Octagonal_Shape &y)
  If the upper bound of `*this` *and* `y` *is exact, it is assigned to* `*this` *and* `true` *is returned, otherwise* `false` *is returned.*

- bool integer_upper_bound_assign_if_exact (const Octagonal_Shape &y)
  If the integer upper bound of `*this` *and* `y` *is exact, it is assigned to* `*this` *and* `true` *is returned; otherwise* `false` *is returned.*

- void difference_assign (const Octagonal_Shape &y)
  Assigns to `*this` *the smallest octagon containing the set difference of* `*this` *and* `y`.

- bool simplify_using_context_assign (const Octagonal_Shape &y)
  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`. *If* `false` *is returned, then the intersection is empty.*

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())
  Assigns to `*this` *the affine image of* `*this` *under the function mapping variable* `var` *into the affine expression specified by* `expr` *and* `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())
  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *into the affine expression specified by* `expr` *and* `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine transfer function* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)
  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine transfer function* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_-Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_-Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the preimage of* `*this` *with respect to the affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)
  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())
  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

- void time_elapse_assign (const Octagonal_Shape &y)
  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`.

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)
  Wraps the specified dimensions of the vector space.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)
  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)
  Possibly tightens `*this` *by dropping some points with non-integer coordinates for the space dimensions corresponding to* `vars`.

- void topological_closure_assign ()
  Assigns to `*this` *its topological closure.*

- void CC76_extrapolation_assign (const Octagonal_Shape &y, unsigned *tp=0)
  Assigns to `*this` *the result of computing the CC76-extrapolation between* `*this` *and* `y`.

- template<typename Iterator >
  void CC76_extrapolation_assign (const Octagonal_Shape &y, Iterator first, Iterator last, unsigned *tp=0)
  Assigns to `*this` *the result of computing the CC76-extrapolation between* `*this` *and* `y`.

- void BHMZ05_widening_assign (const Octagonal_Shape &y, unsigned *tp=0)
  Assigns to `*this` *the result of computing the BHMZ05-widening between* `*this` *and* `y`.

- void widening_assign (const Octagonal_Shape &y, unsigned *tp=0)
  Same as BHMZ05_widening_assign(y, tp).

- void limited_BHMZ05_extrapolation_assign (const Octagonal_Shape &y, const Constraint_System &cs, unsigned *tp=0)
  Improves the result of the BHMZ05-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `*this`.

- void CC76_narrowing_assign (const Octagonal_Shape &y)
  Restores from `y` *the constraints of* `*this`*, lost by CC76-extrapolation applications.*

- void limited_CC76_extrapolation_assign (const Octagonal_Shape &y, const Constraint_System &cs, unsigned *tp=0)
  Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `*this`.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)

Adds `m` new dimensions and embeds the old OS into the new space.

- void add_space_dimensions_and_project (dimension_type m)
  Adds `m` new dimensions to the OS and does not embed it in the new space.
- void concatenate_assign (const Octagonal_Shape &y)
  Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.
- void remove_space_dimensions (const Variables_Set &vars)
  Removes all the specified dimensions.
- void remove_higher_space_dimensions (dimension_type new_dimension)
  Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.
- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)
  Remaps the dimensions of the vector space according to a partial function.
- void expand_space_dimension (Variable var, dimension_type m)
  Creates `m` copies of the space dimension corresponding to `var`.
- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` into `dest`.
- template<typename Interval_Info >
  void refine_fp_interval_abstract_store (Box< Interval< T, Interval_Info > > &store) const
  Refines `store` with the constraints defining `*this`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension that an OS can handle.
- static bool can_recycle_constraint_systems ()
  Returns false indicating that this domain cannot recycle constraints.
- static bool can_recycle_congruence_systems ()
  Returns false indicating that this domain cannot recycle congruences.

Private Types

- typedef Checked_Number< T, Debug_WRD_Extended_Number_Policy > N
  The (extended) numeric type of the inhomogeneous term of the inequalities defining an OS.

Private Member Functions

- bool marked_zero_dim_univ () const
  Returns `true` if the OS is the zero-dimensional universe.

- bool marked_empty () const

  Returns `true` *if the OS is known to be empty.*

- bool marked_strongly_closed () const

  Returns `true` *if* `this->matrix` *is known to be strongly closed.*

- void set_zero_dim_univ ()

  Turns `*this` *into a zero-dimensional universe OS.*

- void set_empty ()

  Turns `*this` *into an empty OS.*

- void set_strongly_closed ()

  Marks `*this` *as strongly closed.*

- void reset_strongly_closed ()

  Marks `*this` *as possibly not strongly closed.*

- N & matrix_at (dimension_type i, dimension_type j)
- const N & matrix_at (dimension_type i, dimension_type j) const
- void refine_no_check (const Constraint &c)

  Uses the constraint `c` *to refine* `*this`.

- void refine_no_check (const Congruence &cg)

  Uses the congruence `cg` *to refine* `*this`.

- void add_octagonal_constraint (dimension_type i, dimension_type j, const N &k)

  Adds the constraint `matrix[i][j] <= k`.

- void add_octagonal_constraint (dimension_type i, dimension_type j, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den)

  Adds the constraint `matrix[i][j] <= num/den`.

- void refine (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Adds to the Octagonal_Shape the constraint var $\bowtie \frac{\text{expr}}{\text{denominator}}$.

- void forget_all_octagonal_constraints (dimension_type v_id)

  Removes all the constraints on variable `v_id`.

- void forget_binary_octagonal_constraints (dimension_type v_id)

  Removes all binary constraints on variable `v_id`.

- void deduce_v_pm_u_bounds (dimension_type v_id, dimension_type last_id, const Linear_Expression &sc_expr, Coefficient_traits::const_reference sc_den, const N &ub_v)

  An helper function for the computation of affine relations.

- void deduce_minus_v_pm_u_bounds (dimension_type v, dimension_type last_v, const Linear_Expression &sc_expr, Coefficient_traits::const_reference sc_den, const N &minus_lb_v)

  An helper function for the computation of affine relations.

- void get_limiting_octagon (const Constraint_System &cs, Octagonal_Shape &limiting_octagon) const

  Adds to `limiting_octagon` *the octagonal differences in* `cs` *that are satisfied by* `*this`.

- void compute_successors (std::vector< dimension_type > &successor) const

  Compute the (zero-equivalence classes) successor relation.

- void compute_leaders (std::vector< dimension_type > &successor, std::vector< dimension_type > &no_sing_leaders, bool &exist_sing_class, dimension_type &sing_leader) const

  Compute the leaders of zero-equivalence classes.

- void compute_leaders (std::vector< dimension_type > &leaders) const

  Compute the leaders of zero-equivalence classes.

- void non_redundant_matrix_entries (std::vector< Bit_Row > &non_redundant) const

  Stores into `non_redundant` *information about the matrix entries that are non-redundant (i.e., will occur in strongly reduced matrix).*

- void strong_reduction_assign () const

  Removes the redundant constraints from `this->matrix`.

- bool is_strongly_reduced () const

  Returns `true` *if and only if* `this->matrix` *is strongly reduced.*

- bool is_strong_coherent () const

  Returns `true` *if in the octagon taken two at a time unary constraints, there is also the constraint that represent their sum.*

- bool tight_coherence_would_make_empty () const
- void strong_closure_assign () const

  Assigns to `this->matrix` *its strong closure.*

- void strong_coherence_assign ()

  Applies the strong-coherence step to `this->matrix`.

- void tight_closure_assign ()

  Assigns to `this->matrix` *its tight closure.*

- void incremental_strong_closure_assign (Variable var) const

  Incrementally computes strong closure, assuming that only constraints affecting variable `var` *need to be considered.*

- bool bounds (const Linear_Expression &expr, bool from_above) const

  Checks if and how `expr` *is bounded in* `*this`.

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included) const

  Maximizes or minimizes `expr` *subject to* `*this`.

- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included, Generator &g) const

  Maximizes or minimizes `expr` *subject to* `*this`.

- void drop_some_non_integer_points_helper (N &elem)

Exception Throwers

- void throw_dimension_incompatible (const char *method, const Octagonal_Shape &x) const
- void throw_dimension_incompatible (const char *method, dimension_type required_dim) const
- void throw_dimension_incompatible (const char *method, const Constraint &c) const
- void throw_dimension_incompatible (const char *method, const Congruence &cg) const
- void throw_dimension_incompatible (const char *method, const Generator &g) const
- void throw_dimension_incompatible (const char *method, const char *name_row, const Linear_Expression &y) const
- void throw_constraint_incompatible (const char *method) const
- void throw_expression_too_complex (const char *method, const Linear_Expression &e) const
- void throw_generic (const char *method, const char *reason) const

Private Attributes

- OR_Matrix< N > matrix

  The matrix that represents the octagonal shape.

- dimension_type space_dim

  Dimension of the space of the octagonal shape.

- Status status

  The status flags to keep track of the internal state.

Static Private Attributes

- static T default_stop_points [ ]

Friends

- class Parma_Polyhedra_Library::Octagonal_Shape
- class Parma_Polyhedra_Library::Box
- bool operator== (const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same octagon.*

- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< U > &x, const Octagonal_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)
- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< U > &x, const Octagonal_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

- template<typename Temp , typename To , typename U >
  bool Parma_Polyhedra_Library::l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< U > &x, const Octagonal_Shape< U > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)
- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Octagonal_Shape< T > &c)

Related Functions

(Note that these are not member functions.)

- template<typename T >
  std::ostream & operator<< (std::ostream &s, const Octagonal_Shape< T > &oct)

  Output operator.

- template<typename T >
  bool operator!= (const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are different shapes.*

- template<typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`*.*

- template<typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the euclidean distance between `x` *and* `y`*.*

- template<typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir)

  Computes the L_∞ *distance between* `x` *and* `y`*.*

- template<typename Temp , typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const Octagonal_Shape< T > &x, const Octagonal_Shape< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ *distance between* `x` *and* `y`*.*

- bool extract_octagonal_difference (const Constraint &c, dimension_type c_space_dim, dimension_type &c_num_vars, dimension_type &c_first_var, dimension_type &c_second_var, Coefficient &c_coeff, Coefficient &c_term)

  Decodes the constraint `c` as an octagonal difference.

- template<typename T >
  void swap (Parma_Polyhedra_Library::Octagonal_Shape< T > &x, Parma_Polyhedra_Library::Octagonal_Shape< T > &y)

  Specializes `std::swap`.

- dimension_type coherent_index (const dimension_type i)

  Returns the index coherent to `i`.

11.149.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::Octagonal_Shape< T >

An octagonal shape. The class template Octagonal_Shape<T> allows for the efficient representation of a restricted kind of *topologically closed* convex polyhedra called *octagonal shapes* (OSs, for short). The name comes from the fact that, in a vector space of dimension 2, bounded OSs are polygons with at most eight sides. The closed affine half-spaces that characterize the OS can be expressed by constraints of the form

$$ax_i + bx_j \leq k$$

where $a, b \in \{-1, 0, 1\}$ and k is a rational number, which are called *octagonal constraints*.

Based on the class template type parameter `T`, a family of extended numbers is built and used to approximate the inhomogeneous term of octagonal constraints. These extended numbers provide a representation for the value $+\infty$, as well as *rounding-aware* implementations for several arithmetic functions. The value of the type parameter `T` may be one of the following:

- a bounded precision integer type (e.g., `int32_t` or `int64_t`);
- a bounded precision floating point type (e.g., `float` or `double`);
- an unbounded integer or rational type, as provided by GMP (i.e., `mpz_class` or `mpq_class`).

The user interface for OSs is meant to be as similar as possible to the one developed for the polyhedron class C_Polyhedron.

The OS domain *optimally supports*:

- tautological and inconsistent constraints and congruences;
- octagonal constraints;
- non-proper congruences (i.e., equalities) that are expressible as octagonal constraints.

Depending on the method, using a constraint or congruence that is not optimally supported by the domain will either raise an exception or result in a (possibly non-optimal) upward approximation.

A constraint is octagonal if it has the form

$$\pm a_i x_i \pm a_j x_j \bowtie b$$

where $\bowtie \in \{\leq, =, \geq\}$ and a_i, a_j, b are integer coefficients such that $a_i = 0$, or $a_j = 0$, or $a_i = a_j$. The user is warned that the above octagonal Constraint object will be mapped into a *correct* and *optimal* approximation that, depending on the expressive power of the chosen template argument `T`, may loose some precision. Also note that strict constraints are not octagonal.

For instance, a Constraint object encoding $3x + 3y \leq 1$ will be approximated by:

- $x + y \leq 1$, if `T` is a (bounded or unbounded) integer type;
- $x + y \leq \frac{1}{3}$, if `T` is the unbounded rational type `mpq_class`;
- $x + y \leq k$, where $k > \frac{1}{3}$, if `T` is a floating point type (having no exact representation for $\frac{1}{3}$).

On the other hand, depending from the context, a Constraint object encoding $3x - y \leq 1$ will be either upward approximated (e.g., by safely ignoring it) or it will cause an exception.

In the following examples it is assumed that the type argument `T` is one of the possible instances listed above and that variables `x`, `y` and `z` are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds an OS corresponding to a cube in $\mathbb{R}^3$, given as a system of constraints:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
cs.insert(z >= 0);
cs.insert(z <= 3);
Octagonal_Shape<T> oct(cs);
```

In contrast, the following code will raise an exception, since constraints 7, 8, and 9 are not octagonal:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
cs.insert(z >= 0);
cs.insert(z <= 3);
cs.insert(x - 3*y <= 5);     // (7)
cs.insert(x - y + z <= 5);   // (8)
cs.insert(x + y + z <= 5);   // (9)
Octagonal_Shape<T> oct(cs);
```

Definition at line 408 of file Octagonal_Shape.defs.hh.

11.149.2 Member Typedef Documentation

11.149.2.1 template<typename T> typedef N Parma_Polyhedra_Library::Octagonal_Shape< T >::coefficient_type

The (extended) numeric type of the inhomogeneous term of the inequalities defining an OS.

Definition at line 429 of file Octagonal_Shape.defs.hh.

11.149.2.2 template<typename T> typedef T Parma_Polyhedra_Library::Octagonal_Shape< T >::coefficient_type_base

The numeric base type upon which OSs are built.

Definition at line 423 of file Octagonal_Shape.defs.hh.

11.149.2.3 template<typename T> typedef Checked_Number<T, Debug_WRD_Extended_Number_Policy> Parma_Polyhedra_Library::Octagonal_Shape< T >::N [private]

The (extended) numeric type of the inhomogeneous term of the inequalities defining an OS.

Definition at line 415 of file Octagonal_Shape.defs.hh.

11.149.3 Constructor & Destructor Documentation

11.149.3.1 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds an universe or empty OS of the specified space dimension.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the OS;

kind Specifies whether the universe or the empty OS has to be built.

Definition at line 108 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.3.2 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Octagonal_Shape< T > & *x*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 121 of file Octagonal_Shape.inlines.hh.

11.149.3.3 template<typename T > template<typename U > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Octagonal_Shape< U > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a conservative, upward approximation of `y`.

The complexity argument is ignored.

Definition at line 128 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_zero_dim_univ(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ().

11.149.3.4 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Constraint_System & *cs*) [inline, explicit]

Builds an OS from the system of constraints `cs`.

The OS inherits the space dimension of `cs`.

Parameters

cs A system of octagonal constraints.

Exceptions

std::invalid_argument Thrown if `cs` contains a constraint which is not optimally supported by the Octagonal shape domain.

Definition at line 144 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Constraint_System::space_dimension().

11.149.3.5 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Congruence_System & *cgs*) [inline, explicit]

Builds an OS from a system of congruences.

The OS inherits the space dimension of `cgs`

Parameters

cgs A system of congruences.

Exceptions

std::invalid_argument Thrown if `cgs` contains a congruence which is not optimally supported by the Octagonal shape domain.

Definition at line 156 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Congruence_System::space_dimension().

11.149.3.6 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Generator_System & *gs*) [inline, explicit]

Builds an OS from the system of generators `gs`.

Builds the smallest OS containing the polyhedron defined by `gs`. The OS inherits the space dimension of `gs`.

Exceptions

std::invalid_argument Thrown if the system of generators is not empty but has no points.

Definition at line 198 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Generator::LINE, Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic(), and Parma_Polyhedra_Library::Generator::type().

11.149.3.7 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds an OS from the polyhedron `ph`.

Builds an OS containing `ph` using algorithms whose complexity does not exceed the one specified by `complexity`. If `complexity` is `ANY_COMPLEXITY`, then the OS built is the smallest one containing `ph`.

Definition at line 44 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Polyhedron::con_sys, Parma_Polyhedra_Library::Polyhedron::constraints(), Parma_Polyhedra_Library::Polyhedron::constraints_are_minimized(), Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Polyhedron::generators(),

Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date(), Parma_Polyhedra_Library::Polyhedron::has_pending_constraints(), Parma_Polyhedra_Library::Polyhedron::has_something_pending(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::MIP_Problem::is_satisfiable(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Polyhedron::is_universe(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, Parma_Polyhedra_Library::MIP_Problem::optimizing_point(), Parma_Polyhedra_Library::POLYNOMIAL_COMPLEXITY, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::MIP_Problem::set_objective_function(), Parma_Polyhedra_Library::MIP_Problem::set_optimization_mode(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), Parma_Polyhedra_Library::SIMPLEX_COMPLEXITY, Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Polyhedron::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.149.3.8 template<typename T > template<typename Interval > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds an OS out of a box.

The OS inherits the space dimension of the box. The built OS is the most precise OS that includes the box.

Parameters

box The box representing the OS to be built.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 169 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::constraints(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Box< ITV >::space_dimension().

11.149.3.9 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const Grid & *grid*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds an OS that approximates a grid.

The OS inherits the space dimension of the grid. The built OS is the most precise OS that includes the grid.

Parameters

grid The grid used to build the OS.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `grid` exceeds the maximum allowed space dimension.

Definition at line 186 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Grid::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Grid::space_dimension().

11.149.3.10 template<typename T > template<typename U > Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape (const BD_Shape< U > & *bd*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds an OS from a BD shape.

The OS inherits the space dimension of the BD shape. The built OS is the most precise OS that includes the BD shape.

Parameters

bd The BD shape used to build the OS.

complexity This argument is ignored as the algorithm used has polynomial complexity.

Exceptions

std::length_error Thrown if the space dimension of `bd` exceeds the maximum allowed space dimension.

Definition at line 201 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::BD_Shape< T >::space_dimension().

11.149.3.11 template<typename T > Parma_Polyhedra_Library::Octagonal_Shape< T >::~Octagonal_Shape () [inline]

Destructor.

Definition at line 233 of file Octagonal_Shape.inlines.hh.

11.149.4 Member Function Documentation

11.149.4.1 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence (const Congruence & *cg*) [inline]

Adds to `*this` a constraint equivalent to the congruence `cg`.

Parameters

cg The congruence to be added.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible, or `cg` is not optimally supported by the OS domain.

Definition at line 454 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences().

11.149.4.2 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences (const Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs The congruences to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the OS domain.

Definition at line 444 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Congruence_System::begin(), and Parma_Polyhedra_Library::Congruence_System::end().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_recycled_congruences(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.4.3 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint (const Constraint & *c*) [inline]

Adds a copy of constraint c to the system of constraints defining *this.

Parameters

c The constraint to be added.

Exceptions

std::invalid_argument Thrown if *this and constraint c are dimension-incompatible, or c is not optimally supported by the OS domain.

Definition at line 372 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign().

11.149.4.4 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints (const Constraint_System & *cs*) [inline]

Adds the constraints in cs to the system of constraints defining *this.

Parameters

cs The constraints that will be added.

Exceptions

std::invalid_argument Thrown if *this and cs are dimension-incompatible, or cs contains a constraint which is not optimally supported by the OS domain.

Definition at line 424 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), and Parma_Polyhedra_Library::Constraint_System::end().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_recycled_constraints(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.4.5 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint (dimension_type *i*, dimension_type *j*, Coefficient_traits::const_reference *num*, Coefficient_traits::const_reference *den*) [inline, private]

Adds the constraint `matrix[i][j] <= num/den`.

Definition at line 406 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::div_round_up(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_size().

11.149.4.6 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint (dimension_type *i*, dimension_type *j*, const N & *k*) [inline, private]

Adds the constraint `matrix[i][j] <= k`.

Definition at line 386 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_size(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::refine().

11.149.4.7 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_recycled_congruences (Congruence_System & *cgs*) [inline]

Adds to `*this` constraints equivalent to the congruences in `cgs`.

Parameters

cgs The congruence system to be added to `*this`. The congruences in `cgs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, or `cgs` contains a congruence which is not optimally supported by the OS domain.

Warning

The only assumption that can be made on `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 438 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruences().

11.149.4.8 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_recycled_constraints (Constraint_System & *cs*) [inline]

Adds the constraints in `cs` to the system of constraints of `*this`.

Parameters

cs The constraint system to be added to `*this`. The constraints in `cs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible, or `cs` contains a constraint which is not optimally supported by the OS domain.

Warning

The only assumption that can be made on `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 432 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraints().

11.149.4.9 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed (dimension_type *m*) [inline]

Adds `m` new dimensions and embeds the old OS into the new space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new OS, which is characterized by a system of constraints in which the variables running through the new dimensions are not constrained. For instance, when starting from the OS $\mathcal{O} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the OS

$$\left\{ (x, y, z)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{O} \right\}.$$

Definition at line 2918 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image().

11.149.4.10 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project (dimension_type *m*) [inline]

Adds m new dimensions to the OS and does not embed it in the new space.

Parameters

m The number of dimensions to add.

The new dimensions will be those having the highest indexes in the new OS, which is characterized by a system of constraints in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the OS $\mathcal{O} \subseteq \mathbb{R}^2$ and adding a third dimension, the result will be the OS

$$\left\{ (x, y, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{O} \right\}.$$

Definition at line 2940 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_end().

11.149.4.11 template<typename T > dimension_type Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension () const [inline]

Returns 0, if *this is empty; otherwise, returns the affine dimension of *this.

Definition at line 551 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::is_discrete().

11.149.4.12 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to *this the affine image of *this under the function mapping variable var into the affine expression specified by expr and denominator.

Parameters

var The variable to which the affine expression is assigned.

expr The numerator of the affine expression.

denominator The denominator of the affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this`.

Definition at line 4417 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image().

11.149.4.13 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` into the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted.

expr The numerator of the affine expression.

denominator The denominator of the affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this`.

Definition at line 4805 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.14 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 7053 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

11.149.4.15 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.149.4.16 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 7066 of file Octagonal_Shape.templates.hh.

11.149.4.17 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign (const Octagonal_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the BHMZ05-widening between `*this` and `y`.

Parameters

y An OS that *must* be contained in `*this`.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3373 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::widening_assign().

11.149.4.18 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 5591 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(),

Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.19 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 6121 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.20 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds (const Linear_Expression & *expr*, bool *from_above*) const [inline, private]

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

from_above `true` if and only if the boundedness of interest is "from above".

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 1147 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds_from_above(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds_from_below().

11.149.4.21 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from above in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 265 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds().

11.149.4.22 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from below in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 271 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds().

11.149.4.23 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::can_recycle_congruence_systems () [inline, static]

Returns false indicating that this domain cannot recycle congruences.

Definition at line 525 of file Octagonal_Shape.inlines.hh.

11.149.4.24 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::can_recycle_constraint_systems () [inline, static]

Returns false indicating that this domain cannot recycle constraints.

Definition at line 519 of file Octagonal_Shape.inlines.hh.

11.149.4.25 template<typename T > template<typename Iterator > void Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign (const Octagonal_Shape< T > & *y*, Iterator *first*, Iterator *last*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the CC76-extrapolation between `*this` and `y`.

Parameters

y An OS that *must* be contained in `*this`.

first An iterator that points to the first stop_point.

last An iterator that points to the last stop_point.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3172 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix<

T >::element_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_-Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_-Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.26 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign (const Octagonal_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Assigns to `*this` the result of computing the CC76-extrapolation between `*this` and `y`.

Parameters

y An OS that *must* be contained in `*this`.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 578 of file Octagonal_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign().

11.149.4.27 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign (const Octagonal_Shape< T > & *y*) [inline]

Restores from `y` the constraints of `*this`, lost by CC76-extrapolation applications.

Parameters

y An OS that *must* contain `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3485 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_-Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_-end(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_-Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_-Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_-dimension_incompatible().

11.149.4.28 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders (std::vector< dimension_type > & *leaders*) const [inline, private]

Compute the leaders of zero-equivalence classes.

It is assumed that the OS is not empty and strongly closed.

Definition at line 2349 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse().

11.149.4.29 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders (std::vector< dimension_type > & *successor*, std::vector< dimension_type > & *no_sing_leaders*, bool & *exist_sing_class*, dimension_type & *sing_leader*) const [inline, private]

Compute the leaders of zero-equivalence classes.

It is assumed that the OS is not empty and strongly closed.

Definition at line 2381 of file Octagonal_Shape.templates.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign().

11.149.4.30 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_successors (std::vector< dimension_type > & *successor*) const [inline, private]

Compute the (zero-equivalence classes) successor relation.

It is assumed that the octagon is not empty and strongly closed.

Definition at line 2317 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse().

11.149.4.31 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

Exceptions

std::length_error Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 670 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim.

11.149.4.32 template<typename T > Congruence_System Parma_Polyhedra_Library::Octagonal_Shape< T >::congruences () const [inline]

Returns a system of (equality) congruences satisfied by `*this`.

Definition at line 218 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences().

Referenced by Parma_Polyhedra_Library::Grid::Grid().

11.149.4.33 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains (Variable *var*) const [inline]

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 1046 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::OR_Matrix< T >::row_size(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.34 template<typename T > Constraint_System Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints () const [inline]

Returns the system of constraints defining `*this`.

Definition at line 6189 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_-

Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_-Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_-end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_-Library::Constraint_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign().

11.149.4.35 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::contains (const Octagonal_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 714 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_-Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_-Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_-incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_-BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_-extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_-assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strictly_contains().

11.149.4.36 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point () const [inline]

Returns `true` if and only if `*this` contains (at least) an integer point.

Definition at line 846 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_-Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::set_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_-

dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty().

11.149.4.37 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds (dimension_type *v*, dimension_type *last_v*, const Linear_Expression & *sc_expr*, Coefficient_traits::const_reference *sc_den*, const N & *minus_lb_v*) [inline, private]

An helper function for the computation of affine relations.

For each variable index `u_id` (less than or equal to `last_id` and different from `v_id`), deduce constraints of the form `-v + u <= k` and `-v - u <= k`, starting from `minus_lb_v`, which is the negation of a lower bound for `v` computed according to `sc_expr` and `sc_den`.

Strong-closure will be able to deduce the constraints `-v - u <= -lb_v - lb_u` and `-v + u <= -lb_v + ub_u`. We can be more precise if variable `u` played an active role in the computation of (the negation of) the lower bound for `v`.

Namely, if the corresponding coefficient `q == sc_expr[u]/sc_den` of `u` in `sc_expr` is greater than zero, we can improve the bound for `-v + u`. In particular:

- if `q >= 1`, then `-v + u <= -lb_v + lb_u`;
- if `0 < q < 1`, then `-v + u <= -lb_v + (q*lb_u + (1-q)*ub_u)`.

Conversely, if `q` is less than zero, we can improve the bound for `-v - u`. In particular:

- if `q <= -1`, then `-v - u <= -lb_v - ub_u`;
- if `-1 < q < 0`, then `-v - u <= -lb_v - ((-q)*ub_u + (1+q)*lb_u)`.

Definition at line 3656 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::neg_assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::refine().

11.149.4.38 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds (dimension_type *v_id*, dimension_type *last_id*, const Linear_Expression & *sc_expr*, Coefficient_traits::const_reference *sc_den*, const N & *ub_v*) [inline, private]

An helper function for the computation of affine relations.

For each variable index `u_id` (less than or equal to `last_id` and different from `v_id`), deduce constraints of the form `v - u <= k` and `v + u <= k`, starting from `ub_v`, which is an upper bound for `v` computed according to `sc_expr` and `sc_den`.

Strong-closure will be able to deduce the constraints `v - u <= ub_v - lb_u` and `v + u <= ub_v + ub_u`. We can be more precise if variable `u` played an active role in the computation of the upper bound for `v`.

Namely, if the corresponding coefficient `q == sc_expr[u]/sc_den` of `u` in `sc_expr` is greater than zero, we can improve the bound for `v - u`. In particular:

- if `q >= 1`, then `v - u <= ub_v - ub_u`;
- if `0 < q < 1`, then `v - u <= ub_v - (q*ub_u + (1-q)*lb_u)`.

Conversely, if `q` is less than zero, we can improve the bound for `v + u`. In particular:

- if `q <= -1`, then `v + u <= ub_v + lb_u`;
- if `-1 < q < 0`, then `v + u <= ub_v + ((-q)*lb_u + (1+q)*ub_u)`.

Definition at line 3537 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::neg_assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::refine().

11.149.4.39 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` the smallest octagon containing the set difference of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2605 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.149.4.40 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 6820 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::is_even(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), and Parma_Polyhedra_Library::Variables_Set::space_dimension().

11.149.4.41 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 6784 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::is_even(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.4.42 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points_helper (N & *elem*) [inline, private]

Definition at line 825 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), and Parma_Polyhedra_Library::Checked::Result.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points().

11.149.4.43 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension (Variable *var*, dimension_type *m*) [inline]

Creates m copies of the space dimension corresponding to var.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if var does not correspond to a dimension of the vector space.

std::length_error Thrown if adding m new space dimensions would cause the vector space to exceed dimension max_space_dimension().

If *this has space dimension n, with $n > 0$, and var has space dimension $k \leq n$, then the k-th space dimension is expanded to m new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 6295 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::max_space_dimension(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.44 template<typename T > memory_size_type Parma_Polyhedra_Library::Octagonal_Shape< T >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by *this.

Definition at line 7087 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::external_memory_in_bytes(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::total_memory_in_bytes().

11.149.4.45 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*) [inline]

Folds the space dimensions in vars into dest.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 6361 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.46 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints (dimension_type *v_id*) [inline, private]

Removes all the constraints on variable `v_id`.

Definition at line 3773 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::PLUS_INFINITY, and Parma_Polyhedra_Library::OR_Matrix< T >::row_size().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain().

11.149.4.47 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_binary_octagonal_constraints (dimension_type *v_id*) [inline, private]

Removes all binary constraints on variable `v_id`.

Definition at line 3795 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), and Parma_Polyhedra_Library::PLUS_INFINITY.

11.149.4.48 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const [inline]

Returns `true` if and only if there exist a unique value `val` such that `*this` saturates the equality `expr = val`.

Parameters

expr The linear expression for which the frequency is needed;

freq_n If `true` is returned, the value is set to 0; Present for interface compatibility with class Grid, where the frequency can have a non-zero value;

freq_d If `true` is returned, the value is set to 1;

val_n The numerator of `val`;

val_d The denominator of `val`;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `false` is returned, then `freq_n`, `freq_d`, `val_n` and `val_d` are left untouched.

Definition at line 896 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::normalize2(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.49 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine transfer function $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression.

relsym The relation symbol.

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `relsym` is a strict relation symbol.

Definition at line 5403 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

11.149.4.50 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine transfer function $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine transfer function.

relsym The relation symbol.

expr The numerator of the right hand side affine expression.

denominator The denominator of the right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this` or if `relsym` is a strict relation symbol.

Definition at line 4914 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::PLUS_INFINITY, PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image().

11.149.4.51 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `relsym` is a strict relation symbol.

Definition at line 5956 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.149.4.52 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the preimage of `*this` with respect to the affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine transfer function.

relsym The relation symbol.

expr The numerator of the right hand side affine expression.

denominator The denominator of the right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a dimension of `*this` or if `relsym` is a strict relation symbol.

Definition at line 5881 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage().

11.149.4.53 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon (const Constraint_System & *cs*, Octagonal_Shape< T > & *limiting_octagon*) const [inline, private]

Adds to `limiting_octagon` the octagonal differences in `cs` that are satisfied by `*this`.

Definition at line 3245 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), and Parma_Polyhedra_Library::Constraint_System::space_dimension().

11.149.4.54 template<typename T > int32_t Parma_Polyhedra_Library::Octagonal_Shape< T >::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 819 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

11.149.4.55 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign (Variable *var*) const [inline, private]

Incrementally computes strong closure, assuming that only constraints affecting variable `var` need to be considered.

Note

It is assumed that `*this`, which was strongly closed, has only been modified by adding constraints affecting variable `var`. If this assumption is not satisfied, i.e., if a non-redundant constraint not affecting variable `var` has been added, the behavior is undefined. Worst-case complexity is $O(n^2)$.

Definition at line 2176 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix,

Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign().

11.149.4.56 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact (const Octagonal_Shape< T > & *y*) [inline]

If the *integer* upper bound of `*this` and `y` is exact, it is assigned to `*this` and `true` is returned; otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Note

This operator is only available when the class template parameter `T` is bound to an integer datatype. The integer upper bound of two rational OS is the smallest rational OS containing all the integral points in the two arguments. In general, the result is *not* an upper bound for the two input arguments, as it may cut away non-integral portions of the two rational shapes.

Implementation is based on Theorem 6.8 of [BHZ09b].

Definition at line 6603 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::non_redundant_matrix_entries(), PPL_COMPILE_TIME_CHECK, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_size(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.149.4.57 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` the intersection of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 3128 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign().

11.149.4.58 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded () const [inline]

Returns `true` if and only if `*this` is a bounded OS.

Definition at line 824 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.4.59 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_discrete () const [inline]

Returns `true` if and only if `*this` is discrete.

Definition at line 252 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension().

11.149.4.60 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from (const Octagonal_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 748 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(),

Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_-Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_-dimension_incompatible().

11.149.4.61 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty () const [inline]

Returns `true` if and only if `*this` is an empty OS.

Definition at line 258 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), and Parma_-Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_-assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_-Library::Pointset_Powerset< PSET >::Pointset_Powerset(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.62 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent () const [inline, private]

Returns `true` if in the octagon taken two at a time unary constraints, there is also the constraint that represent their sum.

Definition at line 1081 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_-Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_-Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_size().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::OK().

11.149.4.63 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced () const [inline, private]

Returns `true` if and only if `this->matrix` is strongly reduced.

Definition at line 1118 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::matrix, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::OR_-Matrix< T >::row_begin(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_end().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign().

11.149.4.64 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_topologically_closed () const [inline]

Returns `true` if and only if `*this` is a topologically closed subset of the vector space.

Definition at line 311 of file Octagonal_Shape.inlines.hh.

11.149.4.65 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe () const [inline]

Returns `true` if and only if `*this` is a universe OS.

Definition at line 801 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min().

11.149.4.66 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_BHMZ05_extrapolation_assign (const Octagonal_Shape< T > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the BHMZ05-widening computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y An OS that *must* be contained in `*this`.

cs The system of constraints used to improve the widened OS.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible or if there is in `cs` a strict inequality.

Definition at line 3438 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.149.4.67 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_extrapolation_assign (const Octagonal_Shape< T > & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0) [inline]

Improves the result of the CC76-extrapolation computation by also enforcing those constraints in `cs` that are satisfied by all the points of `*this`.

Parameters

y An OS that *must* be contained in `*this`.

cs The system of constraints used to improve the widened OS.

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are dimension-incompatible or if `cs` contains a strict inequality.

Definition at line 3326 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

11.149.4.68 template<typename T > template<typename Partial_Function > void Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

Parameters

pfunc The partial function specifying the destiny of each dimension.

The template type parameter Partial_Function must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty codomain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the codomain of the partial function.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned.

The result is undefined if `pfunc` does not encode a partial function with the properties described in the specification of the mapping operator.

Definition at line 3045 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::swap().

11.149.4.69 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty () const [inline, private]

Returns `true` if the OS is known to be empty.

The return value `false` does not necessarily implies that `*this` is non-empty.

Definition at line 78 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::status, and Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_empty().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_-

Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_-with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_-using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.70 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed () const [inline, private]

Returns `true` if `this->matrix` is known to be strongly closed.

The return value `false` does not necessarily implies that `this->matrix` is not strongly closed.

Definition at line 72 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::status, and Parma_Polyhedra_-Library::Octagonal_Shape< T >::Status::test_strongly_closed().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_-narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_-closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_-empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.71 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_zero_dim_univ () const [inline, private]

Returns `true` if the OS is the zero-dimensional universe.

Definition at line 66 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::status, and Parma_Polyhedra_-Library::Octagonal_Shape< T >::Status::test_zero_dim_univ().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.4.72 template<typename T > const Octagonal_Shape< T >::coefficient_type & Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at (dimension_type *i*, dimension_type *j*) const [inline, private]

Definition at line 357 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_size().

11.149.4.73 template<typename T > Octagonal_Shape< T >::coefficient_type & Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at (dimension_type *i*, dimension_type *j*) [inline, private]

Definition at line 368 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_size().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign().

11.149.4.74 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*, Generator & *g*) const [inline, private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in `*` this;

g When maximization or minimization succeeds, will be assigned a point or closure point where `expr` reaches the corresponding extremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `g` are left untouched.

Definition at line 1293 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::MIP_Problem::evaluate_objective_function(), Parma_Polyhedra_Library::Linear_-

Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, Parma_Polyhedra_Library::MIP_Problem::optimizing_point(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.75 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*) const [inline, private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in * this;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d` and `included` are left untouched.

Definition at line 1190 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::MAXIMIZATION, Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::numer_denom(), Parma_Polyhedra_Library::MIP_Problem::optimal_value(), Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM, PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::MIP_Problem::solve(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::maximize(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::minimize().

11.149.4.76 template<typename T > dimension_type Parma_Polyhedra_Library::Octagonal_Shape< T >::max_space_dimension () [inline, static]

Returns the maximum space dimension that an OS can handle.

Definition at line 60 of file Octagonal_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension().

11.149.4.77 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value;

g When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d`, `maximum` and `g` are left untouched.

Definition at line 285 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min().

11.149.4.78 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 277 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with().

11.149.4.79 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value;

g When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

Definition at line 302 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min().

11.149.4.80 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 294 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with().

11.149.4.81 template<typename T > Congruence_System Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences () const [inline]

Returns a minimal system of (equality) congruences satisfied by `*this` with the same affine dimension as `*this`.

Definition at line 583 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Congruence_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::congruences().

11.149.4.82 template<typename T > Constraint_System Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_constraints () const [inline]

Returns a minimized system of constraints defining `*this`.

Definition at line 379 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign().

11.149.4.83 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::non_redundant_matrix_entries (std::vector< Bit_Row > & *non_redundant*) const [inline, private]

Stores into `non_redundant` information about the matrix entries that are non-redundant (i.e., will occur in strongly reduced matrix).

It is assumed that the OS is not empty and strongly closed; moreover, argument `non_redundant` is assumed to be empty.

Definition at line 2448 of file Octagonal_Shape.templates.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.84 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 7093 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::status, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.85 template<typename T > Octagonal_Shape< T > & Parma_Polyhedra_Library::Octagonal_Shape< T >::operator= (const Octagonal_Shape< T > & *y*) [inline]

The assignment operator. (*this and y can be dimension-incompatible.).

Definition at line 224 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

11.149.4.86 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::print () const

Prints *this to std::cerr using operator<<.

11.149.4.87 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline, private]

Adds to the Octagonal_Shape the constraint $\mathrm{var} \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$.

Note that the coefficient of var in expr is null.

Definition at line 3863 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Linear_Expression::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage().

11.149.4.88 template<typename T> template<typename Interval_Info > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_fp_interval_abstract_store (Box< Interval< T, Interval_Info > > & *store*) const [inline]

Refines store with the constraints defining *this.

Parameters

store The interval floating point abstract store to refine.

11.149.4.89 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check (const Congruence & *cg*) [inline, private]

Uses the congruence `cg` to refine `*this`.

Parameters

cg The congruence to be added. Nontrivial proper congruences are ignored. Non-octagonal equalities are ignored.

Warning

If `cg` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 501 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), and Parma_Polyhedra_Library::Congruence::space_dimension().

11.149.4.90 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check (const Constraint & *c*) [inline, private]

Uses the constraint `c` to refine `*this`.

Parameters

c The constraint to be added. Non-octagonal constraints are ignored.

Warning

If `c` and `*this` are dimension-incompatible, the behavior is undefined.

Definition at line 481 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::div_round_up(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraint(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints().

11.149.4.91 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruence (const Congruence & *cg*) [inline]

Uses a copy of congruence `cg` to refine the system of octagonal constraints of `*this`.

Parameters

cg The congruence. If it is not a octagonal equality, it will be ignored.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 476 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.92 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Uses a copy of the congruences in `cgs` to refine the system of octagonal constraints defining `*this`.

Parameters

cgs The congruence system to be used. Congruences that are not octagonal equalities are ignored.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 488 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Congruence_System::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.4.93 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraint (const Constraint & *c*) [inline]

Uses a copy of constraint `c` to refine the system of octagonal constraints defining `*this`.

Parameters

c The constraint. If it is not a octagonal constraint, it will be ignored.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 452 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.94 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints (const Constraint_System & *cs*) [inline]

Uses a copy of the constraints in `cs` to refine the system of octagonal constraints defining `*this`.

Parameters

cs The constraint system to be used. Constraints that are not octagonal are ignored.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 463 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Constraint_System::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.149.4.95 template<typename T > Poly_Gen_Relation Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with (const Generator & *g*) const [inline]

Returns the relations holding between `*this` and the generator `g`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible.

Definition at line 1678 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Coefficient_zero(), Parma_Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.96 template<typename T > Poly_Con_Relation Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with (const Congruence & *cg*) const [inline]

Returns the relations holding between `*this` and the congruence `cg`.

Exceptions

std::invalid_argument Thrown if `*this` and `cg` are dimension-incompatible.

Definition at line 1339 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::maximize(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimize(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.97 template<typename T > Poly_Con_Relation Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with (const Constraint & *c*) const [inline]

Returns the relations holding between `*this` and the constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 1429 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::extract_octagonal_difference(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Constraint::is_nonstrict_inequality(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::maximize(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimize(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::numer_denom(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with().

11.149.4.98 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_higher_space_dimensions (dimension_type *new_dimension*) [inline]

Removes the higher dimensions so that the resulting space will have dimension `new_dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimension` is greater than the space dimension of `*this`.

Definition at line 532 of file Octagonal_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions().

11.149.4.99 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions (const Variables_Set & *vars*) [inline]

Removes all the specified dimensions.

Parameters

vars The set of Variable objects corresponding to the dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 2968 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ(), Parma_Polyhedra_Library::OR_Matrix< T >::shrink(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions().

11.149.4.100 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed () [inline, private]

Marks `*this` as possibly not strongly closed.

Definition at line 102 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::reset_strongly_closed(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points_helper(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign().

11.149.4.101 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty () [inline, private]

Turns `*this` into an empty OS.

Definition at line 90 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_

assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign().

11.149.4.102 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed () [inline, private]

Marks `*this` as strongly closed.

Definition at line 96 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_strongly_closed(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.4.103 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ () [inline, private]

Turns `*this` into a zero-dimensional universe OS.

Definition at line 84 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_zero_dim_univ(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign().

11.149.4.104 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 2669 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::compute_leaders(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_-

Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::is_additive_-inverse(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_-Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::matrix_at(), Parma_Polyhedra_Library::Octagonal_Shape< T >::non_-redundant_matrix_entries(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_-closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_-Library::UNIVERSE.

11.149.4.105 template<typename T > dimension_type Parma_Polyhedra_-Library::Octagonal_Shape< T >::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 246 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_-Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::hash_code(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_-assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::refine_with_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_-with_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::tight_coherence_would_make_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.106 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::strictly_contains (const Octagonal_Shape< T > & *y*) const [inline]

Returns `true` if and only if `*this` strictly contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 611 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::contains().

11.149.4.107 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign () const [inline, private]

Assigns to `this->matrix` its strong closure.

Strong closure is a necessary condition for the precision and/or the correctness of many methods. It explicitly records into `matrix` those constraints that are implicitly obtainable by the other ones, therefore obtaining a canonical representation for the OS.

Definition at line 1953 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.149.4.108 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign () [inline, private]

Applies the strong-coherence step to `this->matrix`.

Definition at line 2094 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::min_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), and Parma_Polyhedra_Library::OR_Matrix< T >::row_end().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign().

11.149.4.109 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign () const [inline, private]

Removes the redundant constraints from `this->matrix`.

Definition at line 2409 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::non_redundant_matrix_entries(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_constraints().

11.149.4.110 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::swap (Octagonal_Shape< T > & *y*) [inline]

Swaps `*this` with octagon `y`. (`*this` and `y` can be dimension-incompatible.).

Definition at line 238 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::status.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.111 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_constraint_incompatible (const char * *method*) const [inline, private]

Definition at line 7236 of file Octagonal_Shape.templates.hh.

11.149.4.112 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, const char * *name_row*, const Linear_Expression & *y*) const [inline, private]

Definition at line 7258 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension().

11.149.4.113 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, const Generator & *g*) const [inline, private]

Definition at line 7225 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Generator::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

11.149.4.114 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, const Congruence & *cg*) const [inline, private]

Definition at line 7214 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

11.149.4.115 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, const Constraint & *c*) const [inline, private]

Definition at line 7203 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Constraint::space_dimension(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

11.149.4.116 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, dimension_type *required_dim*) const [inline, private]

Definition at line 7192 of file Octagonal_Shape.templates.hh.

11.149.4.117 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible (const char * *method*, const Octagonal_Shape< T > & *x*) const [inline, private]

Definition at line 7180 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::simplify_using_context_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.118 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_expression_too_complex (const char * *method*, const Linear_Expression & *e*) const [inline, private]

Definition at line 7246 of file Octagonal_Shape.templates.hh.

11.149.4.119 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_generic (const char * *method*, const char * *reason*) const [inline, private]

Definition at line 7271 of file Octagonal_Shape.templates.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_congruence(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_congruences(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_with_constraints().

11.149.4.120 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign () [inline, private]

Assigns to `this->matrix` its tight closure.

Note

This is *not* marked as a `const` method, as it may modify the rational-valued geometric shape by cutting away non-integral points. The method is only available if the template parameter `T` is bound to an integer datatype.

Definition at line 2142 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::is_even(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), PPL_COMPILE_TIME_CHECK, Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact().

11.149.4.121 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty () const [inline, private]

Definition at line 2123 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse(), Parma_Polyhedra_Library::is_even(), Parma_Polyhedra_Library::is_integer(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_-

Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_-dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), and Parma_-Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign().

11.149.4.122 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 596 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_-dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_-Library::Polyhedron::time_elapse_assign().

11.149.4.123 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 317 of file Octagonal_Shape.inlines.hh.

11.149.4.124 template<typename T > memory_size_type Parma_Polyhedra_-Library::Octagonal_Shape< T >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 813 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::external_memory_in_bytes().

11.149.4.125 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain (const Variables_Set & *vars*) [inline]

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 3836 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.126 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain (Variable *var*) [inline]

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 3816 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

11.149.4.127 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign (const Octagonal_Shape< T > & *y*) [inline]

Assigns to `*this` the smallest OS that contains the convex union of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 2577 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::max_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.4.128 template<typename T > bool Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact (const Octagonal_Shape< T > & *y*) [inline]

If the upper bound of `*this` and `y` is exact, it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Implementation is based on Theorem 6.3 of [BHZ09b].

Definition at line 6450 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::coherent_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::Octagonal_Shape< T >::non_redundant_matrix_entries(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::throw_dimension_incompatible(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.149.4.129 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::widening_assign (const Octagonal_Shape< T > & *y*, unsigned * *tp* = 0) [inline]

Same as BHMZ05_widening_assign(y, tp).

Definition at line 572 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign().

11.149.4.130 template<typename T > void Parma_Polyhedra_Library::Octagonal_Shape< T >::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true) [inline]

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system whose variables are contained in `vars`. If `*pcs` depends on variables not in `vars`, the behavior is undefined. When non-null, the pointed-to constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in `*pcs`.

complexity_threshold A precision parameter of the wrapping operator: higher values result in possibly improved precision.

wrap_individually `true` if the dimensions should be wrapped individually (something that results in much greater efficiency to the detriment of precision).

Exceptions

std::invalid_argument Thrown if `*pcs` is dimension-incompatible with `vars`, or if `*this` is dimension-incompatible `vars` or with `*pcs`.

Definition at line 557 of file Octagonal_Shape.inlines.hh.

11.149.5 Friends And Related Function Documentation

11.149.5.1 template<typename T> dimension_type coherent_index (const dimension_type *i*) [related]

Returns the index coherent to `i`.

Definition at line 50 of file Octagonal_Shape.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at(), Parma_Polyhedra_Library::Octagonal_Shape< T

>::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.5.2 template<typename Temp , typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 684 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.5.3 template<typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the euclidean distance between `x` and `y`.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the euclidean distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 725 of file Octagonal_Shape.inlines.hh.

11.149.5.4 template<typename T> bool extract_octagonal_difference (const Constraint & *c*, dimension_type *c_space_dim*, dimension_type & *c_num_vars*, dimension_type & *c_first_var*, dimension_type & *c_second_var*, Coefficient & *c_coeff*, Coefficient & *c_term*) [related]

Decodes the constraint `c` as an octagonal difference.

Returns

`true` if the constraint `c` is an octagonal difference; `false` otherwise.

Parameters

c The constraint to be decoded.

c_space_dim The space dimension of the constraint `c` (it is *assumed* to match the actual space dimension of `c`).

c_num_vars If `true` is returned, then it will be set to the number of variables having a non-zero coefficient. The only legal values will therefore be 0, 1 and 2.

c_first_var If `true` is returned and if `c_num_vars` is not set to 0, then it will be set to the index of the first variable having a non-zero coefficient in `c`.

c_second_var If `true` is returned and if `c_num_vars` is set to 2, then it will be set to the index of the second variable having a non-zero coefficient in `c`.

c_coeff If `true` is returned and if `c_num_vars` is not set to 0, then it will be set to the value of the first non-zero coefficient in `c`.

c_term If `true` is returned and if `c_num_vars` is not set to 0, then it will be set to the right value of the inhomogeneous term of `c`.

Definition at line 34 of file Octagonal_Shape.cc.

References Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), and Parma_Polyhedra_Library::Constraint::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with().

11.149.5.5 template<typename Temp , typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 749 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.5.6 template<typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 790 of file Octagonal_Shape.inlines.hh.

11.149.5.7 template<typename T > bool operator!= (const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*) [related]

Returns `true` if and only if `x` and `y` are different shapes.

Note that `x` and `y` may be dimension-incompatible shapes: in this case, the value `true` is returned.

Definition at line 351 of file Octagonal_Shape.inlines.hh.

11.149.5.8 template<typename T > std::ostream & operator<< (std::ostream & *s*, const Octagonal_Shape< T > & *x*) [related]

Output operator.

Writes a textual representation of `oct` on `s`: `false` is written if `oct` is an empty polyhedron; `true` is written if `oct` is a universe polyhedron; a system of constraints defining `oct` is written otherwise, all constraints separated by ", ".

Definition at line 6882 of file Octagonal_Shape.templates.hh.

References Parma_Polyhedra_Library::is_additive_inverse(), and Parma_Polyhedra_Library::is_plus_infinity().

11.149.5.9 template<typename T> bool operator== (const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*) [friend]

Returns `true` if and only if `x` and `y` are the same octagon.

Note that `x` and `y` may be dimension-incompatible shapes: in this case, the value `false` is returned.

Definition at line 323 of file Octagonal_Shape.inlines.hh.

11.149.5.10 template<typename T> friend class Parma_Polyhedra_Library::Box [friend]

Definition at line 1777 of file Octagonal_Shape.defs.hh.

11.149.5.11 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< U > & *x*, const Octagonal_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.149.5.12 template<typename T> std::ostream& Parma_Polyhedra_Library::IO_-Operators::operator<< (std::ostream & *s*, const Octagonal_Shape< T > & *c*) [friend]

11.149.5.13 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< U > & *x*, const Octagonal_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.149.5.14 template<typename T> friend class Parma_Polyhedra_Library::Octagonal_Shape [friend]

Definition at line 1776 of file Octagonal_Shape.defs.hh.

11.149.5.15 template<typename T> template<typename Temp , typename To , typename U > bool Parma_Polyhedra_Library::rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< U > & *x*, const Octagonal_Shape< U > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [friend]

11.149.5.16 template<typename Temp , typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 619 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix, Parma_Polyhedra_Library::PLUS_INFINITY, Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim, and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign().

11.149.5.17 template<typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const Octagonal_Shape< T > & *x*, const Octagonal_Shape< T > & *y*, Rounding_Dir *dir*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<To, Extended_Number_Policy>.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using variables of type Checked_Number<Temp, Extended_Number_Policy>.

Definition at line 660 of file Octagonal_Shape.inlines.hh.

11.149.5.18 template<typename T > void swap (Parma_Polyhedra_Library::Octagonal_Shape< T > & *x*, Parma_Polyhedra_Library::Octagonal_Shape< T > & *y*) [related]

Specializes `std::swap`.

Definition at line 843 of file Octagonal_Shape.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::swap().

11.149.6 Member Data Documentation

11.149.6.1 template<typename T> T Parma_Polyhedra_Library::Octagonal_Shape< T >::default_stop_points[] [static, private]

Definition at line 2162 of file Octagonal_Shape.defs.hh.

11.149.6.2 template<typename T> OR_Matrix<N> Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix [private]

The matrix that represents the octagonal shape.

Definition at line 1780 of file Octagonal_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_

dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::operator=(), Parma_Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.149.6.3 template<typename T> dimension_type Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dim [private]

Dimension of the space of the octagonal shape.

Definition at line 1783 of file Octagonal_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Octagonal_Shape< T >::euclidean_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::Octagonal_Shape< T >::l_infinity_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_BHMZ05_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::limited_CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::operator=(), Parma_Polyhedra_Library::Octagonal_Shape< T >::rectilinear_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T

>::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::swap(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::tight_coherence_would_make_empty(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.149.6.4 template<typename T> Status Parma_Polyhedra_Library::Octagonal_Shape< T >::status [private]

The status flags to keep track of the internal state.

Definition at line 1796 of file Octagonal_Shape.defs.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_zero_dim_univ(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_Shape< T >::operator=(), Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed(), Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::swap().

The documentation for this class was generated from the following files:

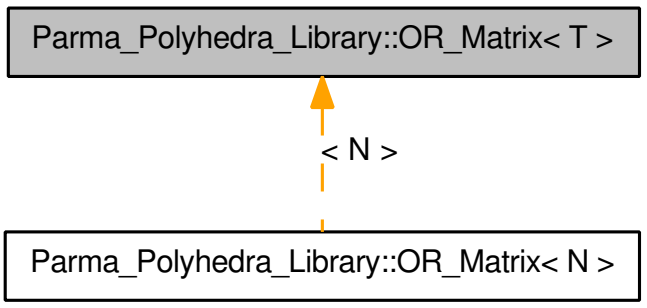
- Octagonal_Shape.defs.hh
- Octagonal_Shape.inlines.hh
- Octagonal_Shape.templates.hh
- Octagonal_Shape.cc

11.150 Parma_Polyhedra_Library::OR_Matrix< T > Class Template Reference

A matrix representing octagonal constraints.

`#include <OR_Matrix.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::OR_Matrix< T >:



Classes

- class any_row_iterator

  A template class to derive both OR_Matrix::iterator and OR_Matrix::const_iterator.

- class Pseudo_Row

  An object that behaves like a matrix's row with respect to the subscript operators.

Public Types

- typedef Pseudo_Row< T > row_reference_type

  A (non const) reference to a matrix's row.

- typedef Pseudo_Row< const T > const_row_reference_type

  A const reference to a matrix's row.

- typedef any_row_iterator< T > row_iterator

  A (non const) row iterator.

- typedef any_row_iterator< const T > const_row_iterator

  A const row iterator.

- typedef DB_Row< T >::iterator element_iterator

  A (non const) element iterator.

- typedef DB_Row< T >::const_iterator const_element_iterator

  A const element iterator.

Public Member Functions

- OR_Matrix (dimension_type space_dim)

  Builds a matrix with specified dimensions.

- OR_Matrix (const OR_Matrix &y)

  Copy constructor.

- template<typename U >
  OR_Matrix (const OR_Matrix< U > &y)

  Constructs a conservative approximation of `y`.

- ~OR_Matrix ()

  Destructor.

- OR_Matrix & operator= (const OR_Matrix &y)

  Assignment operator.

- void swap (OR_Matrix &y)

  Swaps `*this` *with* `y`.

- void grow (dimension_type new_dim)

  Makes the matrix grow by adding more space dimensions.

- void shrink (dimension_type new_dim)

  Makes the matrix shrink by removing the last space dimensions.

- void resize_no_copy (dimension_type new_dim)

  Resizes the matrix without worrying about the old contents.

- dimension_type space_dimension () const

  Returns the space-dimension of the matrix.

- dimension_type num_rows () const

  Returns the number of rows in the matrix.

- row_iterator row_begin ()

  Returns an iterator pointing to the first row, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- row_iterator row_end ()

  Returns the past-the-end const_iterator.

- const_row_iterator row_begin () const

  Returns a const row iterator pointing to the first row, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_row_iterator row_end () const

  Returns the past-the-end const row iterator.

- element_iterator element_begin ()

  Returns an iterator pointing to the first element, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- element_iterator element_end ()

  Returns the past-the-end const_iterator.

- const_element_iterator element_begin () const

  Returns a const element iterator pointing to the first element, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_element_iterator element_end () const

  Returns the past-the-end const element iterator.

- void clear ()

  Clears the matrix deallocating all its rows.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

Subscript operators.

- row_reference_type operator[ ] (dimension_type k)

  Returns a reference to the `k-th` row of the matrix.

- const_row_reference_type operator[ ] (dimension_type k) const

  Returns a constant reference to the `k-th` row of the matrix.

Static Public Member Functions

- static dimension_type max_num_rows ()

  Returns the maximum number of rows of a OR_Matrix.

- static dimension_type row_size (dimension_type k)

  Returns the size of the row of index `k`.

Private Member Functions

- OR_Matrix ()

  Private and not implemented: default construction is not allowed.

Static Private Member Functions

- static dimension_type row_first_element_index (dimension_type k)

  Returns the index into `vec` of the first element of the row of index `k`.

Private Attributes

- DB_Row< T > vec

  Contains the rows of the matrix.

- dimension_type space_dim

  Contains the dimension of the space of the matrix.

- dimension_type vec_capacity

  Contains the capacity of `vec`.

Friends

- class OR_Matrix
- bool operator== (const OR_Matrix< T > &x, const OR_Matrix< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are identical.*

Related Functions

(Note that these are not member functions.)

- template<typename T >
  std::ostream & operator<< (std::ostream &s, const OR_Matrix< T > &m)

  Output operator.

- template<typename T >
  void swap (Parma_Polyhedra_Library::OR_Matrix< T > &x, Parma_Polyhedra_Library::OR_-Matrix< T > &y)

  Specializes `std::swap`.

- template<typename T >
  bool operator!= (const OR_Matrix< T > &x, const OR_Matrix< T > &y)

  Returns `true` *if and only if* `x` *and* `y` *are different.*

- template<typename Temp , typename To , typename T >
  bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const OR_Matrix< T > &x, const OR_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the rectilinear (or Manhattan) distance between `x` *and* `y`.

- template<typename Temp , typename To , typename T >
  bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const OR_Matrix< T > &x, const OR_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the euclidean distance between `x` *and* `y`.

- template<typename Temp , typename To , typename T >
  bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const OR_Matrix< T > &x, const OR_Matrix< T > &y, Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

  Computes the L_∞ distance between x and y.

- template<typename Specialization , typename Temp , typename To , typename T >
  bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > &r, const OR_Matrix< T > &x, const OR_Matrix< T > &y, const Rounding_Dir dir, Temp &tmp0, Temp &tmp1, Temp &tmp2)

11.150.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::OR_Matrix< T >

A matrix representing octagonal constraints. An OR_Matrix object is a DB_Row object that allows the representation of a *pseudo-triangular* matrix, like the following:

```
        _ _
  0    |_|_|
  1    |_|_|_ _
  2    |_|_|_|_|
  3    |_|_|_|_|_ _
  4    |_|_|_|_|_|_|
  5    |_|_|_|_|_|_|
        . . .
        _ _ _ _ _ _         _
2n-2   |_|_|_|_|_|_| ... |_|
2n-1   |_|_|_|_|_|_| ... |_|
        0 1 2 3 4 5  ... 2n-1
```

It is characterized by parameter n that defines the structure, and such that there are 2∗n rows (and 2∗n columns). It provides row_iterators for the access to the rows and element_iterators for the access to the elements.

Definition at line 106 of file OR_Matrix.defs.hh.

11.150.2 Member Typedef Documentation

11.150.2.1 template<typename T> typedef DB_Row<T>::const_iterator Parma_Polyhedra_Library::OR_Matrix< T >::const_element_iterator

A const element iterator.

Definition at line 332 of file OR_Matrix.defs.hh.

11.150.2.2 template<typename T> typedef any_row_iterator<const T> Parma_Polyhedra_Library::OR_Matrix< T >::const_row_iterator

A const row iterator.

Definition at line 326 of file OR_Matrix.defs.hh.

11.150.2.3 template<typename T> typedef Pseudo_Row<const T> Parma_Polyhedra_Library::OR_Matrix< T >::const_row_reference_type

A const reference to a matrix's row.

Definition at line 194 of file OR_Matrix.defs.hh.

11.150.2.4 template<typename T> typedef DB_Row<T>::iterator Parma_Polyhedra_Library::OR_Matrix< T >::element_iterator

A (non const) element iterator.

Definition at line 329 of file OR_Matrix.defs.hh.

11.150.2.5 template<typename T> typedef any_row_iterator<T> Parma_Polyhedra_Library::OR_Matrix< T >::row_iterator

A (non const) row iterator.

Definition at line 323 of file OR_Matrix.defs.hh.

11.150.2.6 template<typename T> typedef Pseudo_Row<T> Parma_Polyhedra_Library::OR_Matrix< T >::row_reference_type

A (non const) reference to a matrix's row.

Definition at line 191 of file OR_Matrix.defs.hh.

11.150.3 Constructor & Destructor Documentation

11.150.3.1 template<typename T > Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix (dimension_type *space_dim*) [inline]

Builds a matrix with specified dimensions.

Parameters

space_dim The space dimension of the matrix that will be created.

This constructor creates a matrix with `2*space_dim` rows. Each element is initialized to plus infinity.

Definition at line 439 of file OR_Matrix.inlines.hh.

11.150.3.2 template<typename T > Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix (const OR_Matrix< T > & *y*) [inline]

Copy constructor.

Definition at line 508 of file OR_Matrix.inlines.hh.

11.150.3.3 template<typename T > template<typename U > Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix (const OR_Matrix< U > & *y*) [inline, explicit]

Constructs a conservative approximation of `y`.

Definition at line 518 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

11.150.3.4 template<typename T > Parma_Polyhedra_Library::OR_Matrix< T >::~OR_Matrix () [inline]

Destructor.

Definition at line 447 of file OR_Matrix.inlines.hh.

11.150.3.5 template<typename T> Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix () [private]

Private and not implemented: default construction is not allowed.

11.150.4 Member Function Documentation

11.150.4.1 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 65 of file OR_Matrix.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), and Parma_Polyhedra_Library::OR_Matrix< T >::space_dimension().

11.150.4.2 template<typename T> void Parma_Polyhedra_Library::OR_Matrix< T >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump().

11.150.4.3 template<typename T > bool Parma_Polyhedra_Library::OR_Matrix< T >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 86 of file OR_Matrix.templates.hh.

References Parma_Polyhedra_Library::is_minus_infinity(), Parma_Polyhedra_Library::Checked::Result, and Parma_Polyhedra_Library::Checked::result_relation().

11.150.4.4 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::clear () [inline]

Clears the matrix deallocating all its rows.

Definition at line 484 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::swap().

11.150.4.5 template<typename T > OR_Matrix< T >::const_element_iterator Parma_Polyhedra_Library::OR_Matrix< T >::element_begin () const [inline]

Returns a const element iterator pointing to the first element, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 389 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::vec.

11.150.4.6 template<typename T > OR_Matrix< T >::element_iterator Parma_Polyhedra_Library::OR_Matrix< T >::element_begin () [inline]

Returns an iterator pointing to the first element, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 377 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::vec.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::OR_-

Matrix< T >::grow(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_reduction_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.150.4.7 template<typename T > OR_Matrix< T >::const_element_iterator Parma_Polyhedra_Library::OR_Matrix< T >::element_end () const [inline]

Returns the past-the-end const element iterator.

Definition at line 395 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::vec.

11.150.4.8 template<typename T > OR_Matrix< T >::element_iterator Parma_Polyhedra_Library::OR_Matrix< T >::element_end () [inline]

Returns the past-the-end const_iterator.

Definition at line 383 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::vec.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::BHMZ05_widening_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_extrapolation_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::CC76_narrowing_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::contains_integer_point(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::Octagonal_Shape< T >::intersection_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_universe(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign().

11.150.4.9 template<typename T > memory_size_type Parma_Polyhedra_Library::OR_Matrix< T >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 33 of file OR_Matrix.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::vec.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::external_memory_in_bytes(), and Parma_Polyhedra_Library::OR_Matrix< T >::total_memory_in_bytes().

11.150.4.10 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::grow (dimension_type *new_dim*) [inline]

Makes the matrix grow by adding more space dimensions.

Parameters

new_dim The new dimension of the resized matrix.

Adds new rows of right dimension to the end if there is enough capacity; otherwise, creates a new matrix, with the specified dimension, copying the old elements in the upper part of the new matrix, which is then assigned to `*this`. Each new element is initialized to plus infinity.

Definition at line 538 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::OR_Matrix< T >::space_dim, Parma_Polyhedra_Library::OR_Matrix< T >::swap(), Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_embed().

11.150.4.11 template<typename T > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::max_num_rows () [inline, static]

Returns the maximum number of rows of a OR_Matrix.

Definition at line 424 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::isqrt().

11.150.4.12 template<typename T > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::num_rows () const [inline]

Returns the number of rows in the matrix.

Definition at line 478 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::space_dimension().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::OR_Matrix< T >::l_m_distance_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::row_end(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.150.4.13 template<typename T > bool Parma_Polyhedra_Library::OR_Matrix< T >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 39 of file OR_Matrix.templates.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::space_dimension(), Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), and Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix().

11.150.4.14 template<typename T > OR_Matrix< T > & Parma_Polyhedra_Library::OR_Matrix< T >::operator= (const OR_Matrix< T > & *y*) [inline]

Assignment operator.

Definition at line 529 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::OR_Matrix< T >::space_dim, Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

11.150.4.15 template<typename T > OR_Matrix< T >::const_row_reference_type Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the `k-th` row of the matrix.

Definition at line 462 of file OR_Matrix.inlines.hh.

References PPL_OR_MATRIX_EXTRA_DEBUG, Parma_Polyhedra_Library::OR_Matrix< T >::row_first_element_index(), Parma_Polyhedra_Library::OR_Matrix< T >::row_size(), and Parma_Polyhedra_Library::OR_Matrix< T >::vec.

11.150.4.16 template<typename T > OR_Matrix< T >::row_reference_type Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the `k-th` row of the matrix.

Definition at line 452 of file OR_Matrix.inlines.hh.

References PPL_OR_MATRIX_EXTRA_DEBUG, Parma_Polyhedra_Library::OR_Matrix< T >::row_first_element_index(), Parma_Polyhedra_Library::OR_Matrix< T >::row_size(), and Parma_Polyhedra_Library::OR_Matrix< T >::vec.

11.150.4.17 template<typename T> void Parma_Polyhedra_Library::OR_Matrix< T >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.150.4.18 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy (dimension_type *new_dim*) [inline]

Resizes the matrix without worrying about the old contents.

Parameters

new_dim The new dimension of the resized matrix.

If the new dimension is greater than the old one, it adds new rows of right dimension to the end if there is enough capacity; otherwise, it creates a new matrix, with the specified dimension, which is then assigned to `*this`. If the new dimension is less than the old one, it erase from the matrix the rows having index greater than 2*new_dim-1

Definition at line 570 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::shrink(), Parma_Polyhedra_Library::OR_Matrix< T >::space_dim, Parma_Polyhedra_Library::OR_Matrix< T >::swap(), Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

11.150.4.19 template<typename T > OR_Matrix< T >::const_row_iterator Parma_Polyhedra_Library::OR_Matrix< T >::row_begin () const [inline]

Returns a const row iterator pointing to the first row, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 365 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::OR_Matrix< T >::vec.

11.150.4.20 template<typename T > OR_Matrix< T >::row_iterator Parma_Polyhedra_Library::OR_Matrix< T >::row_begin () [inline]

Returns an iterator pointing to the first row, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 353 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::OR_Matrix< T >::vec.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::OR_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_

Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_-bounded(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::strong_closure_assign(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::strong_coherence_assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::upper_bound_assign_if_exact().

11.150.4.21 template<typename T > OR_Matrix< T >::const_row_iterator Parma_Polyhedra_Library::OR_Matrix< T >::row_end () const [inline]

Returns the past-the-end const row iterator.

Definition at line 371 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::num_rows().

11.150.4.22 template<typename T > OR_Matrix< T >::row_iterator Parma_Polyhedra_Library::OR_Matrix< T >::row_end () [inline]

Returns the past-the-end const_iterator.

Definition at line 359 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::num_rows().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_space_dimensions_and_-project(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_-Library::OR_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::Octagonal_Shape< T >::concatenate_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_-assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_bounded(), Parma_Polyhedra_-Library::Octagonal_Shape< T >::is_disjoint_from(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strongly_reduced(), Parma_Polyhedra_Library::Octagonal_Shape< T >::map_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::OK(), Parma_Polyhedra_Library::Octagonal_-Shape< T >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_closure_-assign(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::strong_coherence_assign().

11.150.4.23 template<typename T > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::row_first_element_index (dimension_type *k*) [inline, static, private]

Returns the index into `vec` of the first element of the row of index `k`.

Definition at line 40 of file OR_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ]().

11.150.4.24 template<typename T > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::row_size (dimension_type *k*) [inline, static]

Returns the size of the row of index `k`.

Definition at line 46 of file OR_Matrix.inlines.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_octagonal_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::forget_all_octagonal_constraints(), Parma_Polyhedra_Library::Octagonal_Shape< T >::integer_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::is_strong_coherent(), Parma_Polyhedra_Library::Octagonal_Shape< T >::matrix_at(), and Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ]().

11.150.4.25 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::shrink (dimension_type *new_dim*) [inline]

Makes the matrix shrink by removing the last space dimensions.

Parameters

new_dim The new dimension of the resized matrix.

Erases from matrix to the end the rows with index greater than 2∗new_dim-1.

Definition at line 561 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::space_dim, and Parma_Polyhedra_Library::OR_Matrix< T >::vec.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), and Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy().

11.150.4.26 template<typename T > dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::space_dimension () const [inline]

Returns the space-dimension of the matrix.

Definition at line 472 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::space_dim.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::ascii_dump(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::OR_Matrix< T >::OK().

11.150.4.27 template<typename T > void Parma_Polyhedra_Library::OR_Matrix< T >::swap (OR_Matrix< T > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 401 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::space_dim, Parma_Polyhedra_Library::OR_Matrix< T >::vec, and Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::clear(), Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::OR_Matrix< T >::swap().

11.150.4.28 template<typename T > memory_size_type Parma_Polyhedra_Library::OR_Matrix< T >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 433 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::external_memory_in_bytes().

11.150.5 Friends And Related Function Documentation

11.150.5.1 template<typename Temp , typename To , typename T > bool euclidean_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the euclidean distance between `x` and `y`.

If the Euclidean distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 666 of file OR_Matrix.inlines.hh.

11.150.5.2 template<typename Temp , typename To , typename T > bool l_infinity_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the L_∞ distance between `x` and `y`.

If the L_∞ distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 686 of file OR_Matrix.inlines.hh.

11.150.5.3 template<typename Specialization , typename Temp , typename To , typename T > bool l_m_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*, const Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Definition at line 593 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::combine(), Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::finalize(), Parma_Polyhedra_Library::is_plus_infinity(), Parma_Polyhedra_Library::maybe_assign(), Parma_Polyhedra_Library::OR_Matrix< T >::num_rows(), and Parma_Polyhedra_Library::PLUS_INFINITY.

11.150.5.4 template<typename T > bool operator!= (const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 502 of file OR_Matrix.inlines.hh.

11.150.5.5 template<typename T > std::ostream & operator<< (std::ostream & *s*, const OR_Matrix< T > & *m*) [related]

Output operator.

Definition at line 110 of file OR_Matrix.templates.hh.

11.150.5.6 template<typename T> bool operator== (const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*) [friend]

Returns `true` if and only if `x` and `y` are identical.

Definition at line 493 of file OR_Matrix.inlines.hh.

11.150.5.7 template<typename T> friend class OR_Matrix [friend]

Definition at line 362 of file OR_Matrix.defs.hh.

11.150.5.8 template<typename Temp , typename To , typename T > bool rectilinear_distance_assign (Checked_Number< To, Extended_Number_Policy > & *r*, const OR_Matrix< T > & *x*, const OR_Matrix< T > & *y*, Rounding_Dir *dir*, Temp & *tmp0*, Temp & *tmp1*, Temp & *tmp2*) [related]

Computes the rectilinear (or Manhattan) distance between `x` and `y`.

If the rectilinear distance between `x` and `y` is defined, stores an approximation of it into to `r` and returns `true`; returns `false` otherwise.

The direction of the approximation is specified by `dir`.

All computations are performed using the temporary variables `tmp0`, `tmp1` and `tmp2`.

Definition at line 646 of file OR_Matrix.inlines.hh.

11.150.5.9 template<typename T > void swap (Parma_Polyhedra_Library::OR_Matrix< T > & *x*, Parma_Polyhedra_Library::OR_Matrix< T > & *y*) [related]

Specializes `std::swap`.

Definition at line 708 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::swap().

11.150.6 Member Data Documentation

11.150.6.1 template<typename T> dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::space_dim [private]

Contains the dimension of the space of the matrix.

Definition at line 375 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::OR_-Matrix< T >::operator=(), Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy(), Parma_-Polyhedra_Library::OR_Matrix< T >::shrink(), Parma_Polyhedra_Library::OR_Matrix< T >::space_-dimension(), and Parma_Polyhedra_Library::OR_Matrix< T >::swap().

11.150.6.2 template<typename T> DB_Row<T> Parma_Polyhedra_Library::OR_Matrix< T >::vec [private]

Contains the rows of the matrix.

A DB_Row which contains the rows of the OR_Matrix inserting each successive row to the end of the vec. To contain all the elements of OR_Matrix the size of the DB_Row is 2∗n∗(n+1), where the n is the characteristic parameter of OR_Matrix.

Definition at line 372 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::element_begin(), Parma_Polyhedra_-

Library::OR_Matrix< T >::element_end(), Parma_Polyhedra_Library::OR_Matrix< T >::external_memory_in_bytes(), Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::OR_Matrix< T >::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::operator=(), Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ](), Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix(), Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy(), Parma_Polyhedra_Library::OR_Matrix< T >::row_begin(), Parma_Polyhedra_Library::OR_Matrix< T >::shrink(), and Parma_Polyhedra_Library::OR_Matrix< T >::swap().

11.150.6.3 template<typename T> dimension_type Parma_Polyhedra_Library::OR_Matrix< T >::vec_capacity [private]

Contains the capacity of `vec`.

Definition at line 378 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::grow(), Parma_Polyhedra_Library::OR_Matrix< T >::OK(), Parma_Polyhedra_Library::OR_Matrix< T >::operator=(), Parma_Polyhedra_Library::OR_Matrix< T >::OR_Matrix(), Parma_Polyhedra_Library::OR_Matrix< T >::resize_no_copy(), and Parma_Polyhedra_Library::OR_Matrix< T >::swap().

The documentation for this class was generated from the following files:

- OR_Matrix.defs.hh
- OR_Matrix.inlines.hh
- OR_Matrix.templates.hh

11.151 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > Class Template Reference

The partially reduced product of two abstractions.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- Partially_Reduced_Product (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

  Builds an object having the specified properties.

- Partially_Reduced_Product (const Congruence_System &cgs)

  Builds a pair, copying a system of congruences.

- Partially_Reduced_Product (Congruence_System &cgs)

  Builds a pair, recycling a system of congruences.

- Partially_Reduced_Product (const Constraint_System &cs)

  Builds a pair, copying a system of constraints.

- Partially_Reduced_Product (Constraint_System &cs)

  Builds a pair, recycling a system of constraints.

- Partially_Reduced_Product (const C_Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product, from a C polyhedron.

- Partially_Reduced_Product (const NNC_Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product, from an NNC polyhedron.

- Partially_Reduced_Product (const Grid &gr, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product, from a grid.

- template<typename Interval >
  Partially_Reduced_Product (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product out of a box.

- template<typename U >
  Partially_Reduced_Product (const BD_Shape< U > &bd, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product out of a BD shape.

- template<typename U >
  Partially_Reduced_Product (const Octagonal_Shape< U > &os, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a product out of an octagonal shape.

- Partially_Reduced_Product (const Partially_Reduced_Product &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- template<typename E1 , typename E2 , typename S >
  Partially_Reduced_Product (const Partially_Reduced_Product< E1, E2, S > &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a conservative, upward approximation of `y`.

- Partially_Reduced_Product & operator= (const Partially_Reduced_Product &y)

  The assignment operator. (`*this` *and* `y` *can be dimension-incompatible.).*

- bool reduce () const

  Reduce.

Member Functions that Do Not Modify the Partially_Reduced_Product

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- dimension_type affine_dimension () const

  Returns the minimum affine dimension (see also grid affine dimension) of the components of `*this`.

- const D1 & domain1 () const

Returns a constant reference to the first of the pair.

- const D2 & domain2 () const
  Returns a constant reference to the second of the pair.
- Constraint_System constraints () const
  Returns a system of constraints which approximates `*this`.
- Constraint_System minimized_constraints () const
  Returns a system of constraints which approximates `*this`*, in reduced form.*
- Congruence_System congruences () const
  Returns a system of congruences which approximates `*this`.
- Congruence_System minimized_congruences () const
  Returns a system of congruences which approximates `*this`*, in reduced form.*
- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between `*this` *and* `c`.
- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between `*this` *and* `cg`.
- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between `*this` *and* `g`.
- bool is_empty () const
  Returns `true` *if and only if either of the components of* `*this` *are empty.*
- bool is_universe () const
  Returns `true` *if and only if both of the components of* `*this` *are the universe.*
- bool is_topologically_closed () const
  Returns `true` *if and only if both of the components of* `*this` *are topologically closed subsets of the vector space.*
- bool is_disjoint_from (const Partially_Reduced_Product &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are componentwise disjoint.*
- bool is_discrete () const
  Returns `true` *if and only if a component of* `*this` *is discrete.*
- bool is_bounded () const
  Returns `true` *if and only if a component of* `*this` *is bounded.*
- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.
- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded in* `*this`.
- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded in* `*this`.
- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const

Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &point) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below i* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &point) const

  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool contains (const Partially_Reduced_Product &y) const

  Returns `true` *if and only if each component of* `*this` *contains the corresponding component of* `y`*.*

- bool strictly_contains (const Partially_Reduced_Product &y) const

  Returns `true` *if and only if each component of* `*this` *strictly contains the corresponding component of* `y`*.*

- bool OK () const

  Checks if all the invariants are satisfied.

Space Dimension Preserving Member Functions that May Modify the Partially_Reduced_Product

- void add_constraint (const Constraint &c)

  Adds constraint `c` *to* `*this`*.*

- void refine_with_constraint (const Constraint &c)

  Use the constraint `c` *to refine* `*this`*.*

- void add_congruence (const Congruence &cg)

  Adds a copy of congruence `cg` *to* `*this`*.*

- void refine_with_congruence (const Congruence &cg)

  Use the congruence `cg` *to refine* `*this`*.*

- void add_congruences (const Congruence_System &cgs)

  Adds a copy of the congruences in `cgs` *to* `*this`*.*

- void refine_with_congruences (const Congruence_System &cgs)

  Use the congruences in `cgs` *to refine* `*this`*.*

- void add_recycled_congruences (Congruence_System &cgs)

  Adds the congruences in `cgs` *to* *this.

- void add_constraints (const Constraint_System &cs)

  Adds a copy of the constraint system in `cs` *to* `*this`*.*

- void refine_with_constraints (const Constraint_System &cs)

  Use the constraints in `cs` to refine `*this`.

- void add_recycled_constraints (Constraint_System &cs)

  Adds the constraint system in `cs` to `*this`.

- void unconstrain (Variable var)

  Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

- void unconstrain (const Variables_Set &vars)

  Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

- void intersection_assign (const Partially_Reduced_Product &y)

  Assigns to `*this` the componentwise intersection of `*this` and `y`.

- void upper_bound_assign (const Partially_Reduced_Product &y)

  Assigns to `*this` an upper bound of `*this` and `y` computed on the corresponding components.

- bool upper_bound_assign_if_exact (const Partially_Reduced_Product &y)

  Assigns to `*this` an upper bound of `*this` and `y` computed on the corresponding components. If it is exact on each of the components of `*this`, `true` is returned, otherwise `false` is returned.

- void difference_assign (const Partially_Reduced_Product &y)

  Assigns to `*this` an approximation of the set-theoretic difference of `*this` and `y`.

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` the affine image of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym` (see also generalized affine relation.).*

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`. (see also generalized affine relation.).*

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`. (see also generalized affine relation.).*

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\text{lhs}' \bowtie \text{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`. *(see also generalized affine relation.).*

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

- void time_elapse_assign (const Partially_Reduced_Product &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`. *(See also time-elapse.).*

- void topological_closure_assign ()

  Assigns to `*this` *its topological closure.*

- void widening_assign (const Partially_Reduced_Product &y, unsigned *tp=NULL)

  Assigns to `*this` *the result of computing the "widening" between* `*this` *and* `y`.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates for the space dimensions corresponding to* `vars`.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)

  Adds `m` *new space dimensions and embeds the components of* `*this` *in the new vector space.*

- void add_space_dimensions_and_project (dimension_type m)

  Adds `m` *new space dimensions and does not embed the components in the new vector space.*

- void concatenate_assign (const Partially_Reduced_Product &y)

  Assigns to the first (resp., second) component of `*this` *the "concatenation" of the first (resp., second) components of* `*this` *and* `y`, *taken in this order. See also Concatenating Polyhedra.*

- void remove_space_dimensions (const Variables_Set &vars)

  Removes all the specified dimensions from the vector space.

- void remove_higher_space_dimensions (dimension_type new_dimension)

  Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`.

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)

  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)

Creates `m` *copies of the space dimension corresponding to* `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` *into* `dest`.

Miscellaneous Member Functions

- ~Partially_Reduced_Product ()
  Destructor.
- void swap (Partially_Reduced_Product &y)
  Swaps `*this` *with product* `y`. *(*`*this` *and* `y` *can be dimension-incompatible.).*
- void ascii_dump () const
  Writes to `std::cerr` *an ASCII representation of* `*this`.
- void ascii_dump (std::ostream &s) const
  Writes to `s` *an ASCII representation of* `*this`.
- void print () const
  Prints `*this` *to* `std::cerr` *using* `operator<<`.
- bool ascii_load (std::istream &s)
  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*
- memory_size_type total_memory_in_bytes () const
  Returns the total size in bytes of the memory occupied by `*this`.
- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.
- int32_t hash_code () const
  Returns a 32-bit hash code for `*this`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension this product can handle.

Protected Types

- typedef D1 Domain1
  The type of the first component.
- typedef D2 Domain2
  The type of the second component.

Protected Member Functions

- void clear_reduced_flag () const
  Clears the reduced flag.
- void set_reduced_flag () const
  Sets the reduced flag.
- bool is_reduced () const
  Return `true` *if and only if the reduced flag is set.*

Protected Attributes

- D1 d1
  The first component.
- D2 d2
  The second component.
- bool reduced
  Flag to record whether the components are reduced with respect to each other and the reduction class.

Friends

- bool operator== (const Partially_Reduced_Product< D1, D2, R > &x, const Partially_Reduced_Product< D1, D2, R > &y)
  Returns `true` *if and only if the components of* `x` *and* `y` *are pairwise equal.*
- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Partially_Reduced_Product< D1, D2, R > &dp)

Related Functions

(Note that these are not member functions.)

- template<typename D1 , typename D2 , typename R >
  std::ostream & operator<< (std::ostream &s, const Partially_Reduced_Product< D1, D2, R > &dp)
  Output operator.
- template<typename D1 , typename D2 , typename R >
  bool operator!= (const Partially_Reduced_Product< D1, D2, R > &x, const Partially_Reduced_Product< D1, D2, R > &y)
  Returns `true` *if and only if the components of* `x` *and* `y` *are not pairwise equal.*
- template<typename D1 , typename D2 , typename R >
  void swap (Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > &x, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > &y)
  Specializes `std::swap`.

11.151.1 Detailed Description

template<typename D1, typename D2, typename R> class Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >

The partially reduced product of two abstractions.

Warning

At present, the supported instantiations for the two domain templates `D1` and `D2` are the simple pointset domains: `C_Polyhedron`, `NNC_Polyhedron`, `Grid`, `Octagonal_Shape<T>`, `BD_Shape<T>`, `Box<T>`.

An object of the class `Partially_Reduced_Product<D1, D2, R>` represents the (partially reduced) product of two pointset domains `D1` and `D2` where the form of any reduction is defined by the reduction class `R`.

Suppose D_1 and D_2 are two abstract domains with concretization functions: $\gamma_1 \colon D_1 \to \mathbb{R}^n$ and $\gamma_2 \colon D_2 \to \mathbb{R}^n$, respectively.

The partially reduced product $D = D_1 \times D_2$, for any reduction class `R`, has a concretization $\gamma \colon D \to \mathbb{R}^n$ where, if $d = (d_1, d_2) \in D$

$$\gamma(d) = \gamma_1(d_1) \cap \gamma_2(d_2).$$

The operations are defined to be the result of applying the corresponding operations on each of the components provided the product is already reduced by the reduction method defined by `R`. In particular, if `R` is the `No_Reduction<D1, D2>` class, then the class `Partially_Reduced_Product<D1, D2, R>` domain is the direct product as defined in [CC79].

How the results on the components are interpreted and combined depend on the specific test. For example, the test for emptiness will first make sure the product is reduced (using the reduction method provided by `R` if it is not already known to be reduced) and then test if either component is empty; thus, if `R` defines no reduction between its components and $d = (G, P) \in (\mathbb{G} \times \mathbb{P})$ is a direct product in one dimension where G denotes the set of numbers that are integral multiples of 3 while P denotes the set of numbers between 1 and 2, then an operation that tests for emptiness should return false. However, the test for the universe returns true if and only if the test `is_universe()` on both components returns true.

In all the examples it is assumed that the template `R` is the `No_Reduction<D1, D2>` class and that variables `x` and `y` are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a direct product of a Grid and NNC Polyhedron, corresponding to the positive even integer pairs in $\mathbb{R}^2$, given as a system of congruences:

```
Congruence_System cgs;
cgs.insert((x %= 0) / 2);
cgs.insert((y %= 0) / 2);
Partially_Reduced_Product<Grid, NNC_Polyhedron, No_Reduction<D1, D2> >
  dp(cgs);
dp.add_constraint(x >= 0);
dp.add_constraint(y >= 0);
```

Example 2

The following code builds the same product in $\mathbb{R}^2$:

```
Partially_Reduced_Product<Grid, NNC_Polyhedron, No_Reduction<D1, D2> > dp(2);
dp.add_constraint(x >= 0);
dp.add_constraint(y >= 0);
dp.add_congruence((x %= 0) / 2);
dp.add_congruence((y %= 0) / 2);
```

Example 3

The following code will write "dp is empty":

```
Partially_Reduced_Product<Grid, NNC_Polyhedron, No_Reduction<D1, D2> > dp(1);
dp.add_congruence((x %= 0) / 2);
dp.add_congruence((x %= 1) / 2);
if (dp.is_empty())
  cout << "dp is empty." << endl;
else
  cout << "dp is not empty." << endl;
```

Example 4

The following code will write "dp is not empty":

```
Partially_Reduced_Product<Grid, NNC_Polyhedron, No_Reduction<D1, D2> > dp(1);
dp.add_congruence((x %= 0) / 2);
dp.add_constraint(x >= 1);
dp.add_constraint(x <= 1);
if (dp.is_empty())
  cout << "dp is empty." << endl;
else
  cout << "dp is not empty." << endl;
```

Definition at line 412 of file Partially_Reduced_Product.defs.hh.

11.151.2 Member Typedef Documentation

11.151.2.1 template<typename D1, typename D2, typename R> typedef D1 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Domain1 [protected]

The type of the first component.

Definition at line 1617 of file Partially_Reduced_Product.defs.hh.

11.151.2.2 template<typename D1, typename D2, typename R> typedef D2 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Domain2 [protected]

The type of the second component.

Definition at line 1620 of file Partially_Reduced_Product.defs.hh.

11.151.3 Constructor & Destructor Documentation

11.151.3.1 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds an object having the specified properties.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the pair;

kind Specifies whether a universe or an empty pair has to be built.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 46 of file Partially_Reduced_Product.inlines.hh.

11.151.3.2 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Congruence_System & *cgs*) [inline, explicit]

Builds a pair, copying a system of congruences.

The pair inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences to be approximated by the pair.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 56 of file Partially_Reduced_Product.inlines.hh.

11.151.3.3 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (Congruence_System & *cgs*) [inline, explicit]

Builds a pair, recycling a system of congruences.

The pair inherits the space dimension of the congruence system.

Parameters

cgs The system of congruences to be approximates by the pair. Its data-structures may be recycled to build the pair.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 64 of file Partially_Reduced_Product.inlines.hh.

11.151.3.4 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Constraint_System & *cs*) [inline, explicit]

Builds a pair, copying a system of constraints.

The pair inherits the space dimension of the constraint system.

Parameters

cs The system of constraints to be approximated by the pair.

Exceptions

std::length_error Thrown if `num_dimensions` exceeds the maximum allowed space dimension.

Definition at line 72 of file Partially_Reduced_Product.inlines.hh.

11.151.3.5 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (Constraint_System & *cs*) [inline, explicit]

Builds a pair, recycling a system of constraints.

The pair inherits the space dimension of the constraint system.

Parameters

cs The system of constraints to be approximated by the pair.

Exceptions

std::length_error Thrown if the space dimension of `cs` exceeds the maximum allowed space dimension.

Definition at line 80 of file Partially_Reduced_Product.inlines.hh.

11.151.3.6 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const C_Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a product, from a C polyhedron.

Builds a product containing ph using algorithms whose complexity does not exceed the one specified by complexity. If complexity is ANY_COMPLEXITY, then the built product is the smallest one containing ph. The product inherits the space dimension of the polyhedron.

Parameters

ph The polyhedron to be approximated by the product.

complexity The complexity that will not be exceeded.

Exceptions

std::length_error Thrown if the space dimension of ph exceeds the maximum allowed space dimension.

Definition at line 88 of file Partially_Reduced_Product.inlines.hh.

11.151.3.7 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const NNC_Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a product, from an NNC polyhedron.

Builds a product containing ph using algorithms whose complexity does not exceed the one specified by complexity. If complexity is ANY_COMPLEXITY, then the built product is the smallest one containing ph. The product inherits the space dimension of the polyhedron.

Parameters

ph The polyhedron to be approximated by the product.

complexity The complexity that will not be exceeded.

Exceptions

std::length_error Thrown if the space dimension of ph exceeds the maximum allowed space dimension.

Definition at line 97 of file Partially_Reduced_Product.inlines.hh.

11.151.3.8 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Grid & *gr*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a product, from a grid.

Builds a product containing gr. The product inherits the space dimension of the grid.

Parameters

gr The grid to be approximated by the product.

complexity The complexity is ignored.

Exceptions

std::length_error Thrown if the space dimension of `gr` exceeds the maximum allowed space dimension.

Definition at line 106 of file Partially_Reduced_Product.inlines.hh.

11.151.3.9 template<typename D1 , typename D2 , typename R > template<typename Interval > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Builds a product out of a box.

Builds a product containing `box`. The product inherits the space dimension of the box.

Parameters

box The box representing the pair to be built.

complexity The complexity is ignored.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 115 of file Partially_Reduced_Product.inlines.hh.

11.151.3.10 template<typename D1 , typename D2 , typename R > template<typename U > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const BD_Shape< U > & *bd*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Builds a product out of a BD shape.

Builds a product containing `bd`. The product inherits the space dimension of the BD shape.

Parameters

bd The BD shape representing the product to be built.

complexity The complexity is ignored.

Exceptions

std::length_error Thrown if the space dimension of `bd` exceeds the maximum allowed space dimension.

Definition at line 124 of file Partially_Reduced_Product.inlines.hh.

11.151.3.11 template<typename D1 , typename D2 , typename R > template<typename U > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Octagonal_Shape< U > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Builds a product out of an octagonal shape.

Builds a product containing `os`. The product inherits the space dimension of the octagonal shape.

Parameters

os The octagonal shape representing the product to be built.

complexity The complexity is ignored.

Exceptions

std::length_error Thrown if the space dimension of `os` exceeds the maximum allowed space dimension.

Definition at line 133 of file Partially_Reduced_Product.inlines.hh.

11.151.3.12 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Partially_Reduced_Product< D1, D2, R > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

Definition at line 141 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

11.151.3.13 template<typename D1 , typename D2 , typename R > template<typename E1 , typename E2 , typename S > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product (const Partially_Reduced_Product< E1, E2, S > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a conservative, upward approximation of `y`.

The complexity argument is ignored.

Definition at line 151 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain1(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain2(), and Parma_Polyhedra_Library::swap().

11.151.3.14 template<typename D1 , typename D2 , typename R > Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::~Partially_Reduced_Product () [inline]

Destructor.

Definition at line 165 of file Partially_Reduced_Product.inlines.hh.

11.151.4 Member Function Documentation

11.151.4.1 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_congruence (const Congruence & *cg*) [inline]

Adds a copy of congruence `cg` to `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 397 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.2 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_congruences (const Congruence_System & *cgs*) [inline]

Adds a copy of the congruences in `cgs` to `*this`.

Parameters

cgs The congruence system to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 432 of file Partially_Reduced_Product.inlines.hh.

11.151.4.3 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_constraint (const Constraint & *c*) [inline]

Adds constraint `c` to `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible.

Definition at line 381 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.4 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_constraints (const Constraint_System & *cs*) [inline]

Adds a copy of the constraint system in `cs` to `*this`.

Parameters

cs The constraint system to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 414 of file Partially_Reduced_Product.inlines.hh.

11.151.4.5 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_recycled_congruences (Congruence_System & *cgs*) [inline]

Adds the congruences in `cgs` to *this.

Parameters

cgs The congruence system to be added that may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Warning

The only assumption that can be made about `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 115 of file Partially_Reduced_Product.templates.hh.

11.151.4.6 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_recycled_constraints (Constraint_System & *cs*) [inline]

Adds the constraint system in `cs` to `*this`.

Parameters

cs The constraint system to be added that may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Warning

The only assumption that can be made about `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 95 of file Partially_Reduced_Product.templates.hh.

11.151.4.7 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_space_dimensions_and_embed (dimension_type *m*) [inline]

Adds `m` new space dimensions and embeds the components of `*this` in the new vector space.

Parameters

m The number of dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 577 of file Partially_Reduced_Product.inlines.hh.

11.151.4.8 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_space_dimensions_and_project (dimension_type *m*) [inline]

Adds `m` new space dimensions and does not embed the components in the new vector space.

Parameters

m The number of space dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 585 of file Partially_Reduced_Product.inlines.hh.

11.151.4.9 template<typename D1 , typename D2 , typename R > dimension_type Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_dimension () const [inline]

Returns the minimum affine dimension (see also grid affine dimension) of the components of `*this`.

Definition at line 189 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.10 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the affine image of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 263 of file Partially_Reduced_Product.inlines.hh.

11.151.4.11 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 274 of file Partially_Reduced_Product.inlines.hh.

11.151.4.12 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_dump (std::ostream & *s*) const [inline]

Writes to `s` an ASCII representation of `*this`.

Definition at line 696 of file Partially_Reduced_Product.inlines.hh.

11.151.4.13 template<typename D1, typename D2, typename R> void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.151.4.14 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_load (std::istream & *s*) [inline]

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 434 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

11.151.4.15 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 332 of file Partially_Reduced_Product.inlines.hh.

11.151.4.16 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 344 of file Partially_Reduced_Product.inlines.hh.

11.151.4.17 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::bounds_from_above (const Linear_Expression & *expr*) const `[inline]`

Returns `true` if and only if `expr` is bounded in `*this`.

This method is the same as bounds_from_below.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 538 of file Partially_Reduced_Product.inlines.hh.

11.151.4.18 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded in `*this`.

This method is the same as bounds_from_above.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 546 of file Partially_Reduced_Product.inlines.hh.

11.151.4.19 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag () const [inline, protected]

Clears the reduced flag.

Definition at line 682 of file Partially_Reduced_Product.inlines.hh.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_congruence(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_constraint(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_congruence(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_constraint().

11.151.4.20 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::concatenate_assign (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to the first (resp., second) component of `*this` the "concatenation" of the first (resp., second) components of `*this` and `y`, taken in this order. See also Concatenating Polyhedra.

Exceptions

std::length_error Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 593 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_reduced().

11.151.4.21 template<typename D1 , typename D2 , typename R > Congruence_System Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences () const [inline]

Returns a system of congruences which approximates `*this`.

Definition at line 69 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Congruence_System::insert(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.22 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constrains (Variable *var*) const [inline]

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 553 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.23 template<typename D1 , typename D2 , typename R > Constraint_System Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints () const [inline]

Returns a system of constraints which approximates `*this`.

Definition at line 38 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::insert(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.24 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::contains (const Partially_Reduced_Product< D1, D2, R > & *y*) const [inline]

Returns `true` if and only if each component of `*this` contains the corresponding component of `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 645 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.25 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::difference_assign (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to `*this` an approximation of the set-theoretic difference of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 225 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.26 template<typename D1 , typename D2 , typename R > const D1 & Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain1 () const [inline]

Returns a constant reference to the first of the pair.

Definition at line 480 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product().

11.151.4.27 template<typename D1 , typename D2 , typename R > const D2 & Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain2 () const [inline]

Returns a constant reference to the second of the pair.

Definition at line 487 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product().

11.151.4.28 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 460 of file Partially_Reduced_Product.inlines.hh.

11.151.4.29 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 450 of file Partially_Reduced_Product.inlines.hh.

11.151.4.30 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::expand_space_dimension (Variable *var*, dimension_type *m*) [inline]

Creates `m` copies of the space dimension corresponding to `var`.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if `var` does not correspond to a dimension of the vector space.

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

If `*this` has space dimension n, with $n > 0$, and `var` has space dimension $k \leq n$, then the k-th space dimension is expanded to `m` new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 628 of file Partially_Reduced_Product.inlines.hh.

11.151.4.31 template<typename D1 , typename D2 , typename R > memory_size_type Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 170 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::total_memory_in_bytes().

11.151.4.32 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*) [inline]

Folds the space dimensions in `vars` into `dest`.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 636 of file Partially_Reduced_Product.inlines.hh.

11.151.4.33 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`. (see also generalized affine relation.).

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 309 of file Partially_Reduced_Product.inlines.hh.

11.151.4.34 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym` (see also generalized affine relation.).

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 285 of file Partially_Reduced_Product.inlines.hh.

11.151.4.35 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`. (see also generalized affine relation.).

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 320 of file Partially_Reduced_Product.inlines.hh.

11.151.4.36 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`. (see also generalized affine relation.).

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 297 of file Partially_Reduced_Product.inlines.hh.

11.151.4.37 template<typename D1 , typename D2 , typename R > int32_t Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 709 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::space_dimension().

11.151.4.38 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::intersection_assign (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to `*this` the componentwise intersection of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 216 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.39 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_bounded () const [inline]

Returns `true` if and only if a component of `*this` is bounded.

Definition at line 530 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.40 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_discrete () const [inline]

Returns `true` if and only if a component of `*this` is discrete.

Definition at line 523 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.41 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_disjoint_from (const Partially_Reduced_Product< D1, D2, R > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are componentwise disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are dimension-incompatible.

Definition at line 515 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.42 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_empty () const [inline]

Returns `true` if and only if either of the components of `*this` are empty.

Definition at line 494 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset().

11.151.4.43 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_reduced () const [inline, protected]

Return `true` if and only if the reduced flag is set.

Definition at line 676 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::concatenate_assign(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.44 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_topologically_closed () const [inline]

Returns `true` if and only if both of the components of `*this` are topologically closed subsets of the vector space.

Definition at line 507 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.45 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_universe () const [inline]

Returns `true` if and only if both of the components of `*this` are the universe.

Definition at line 501 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.46 template<typename D1 , typename D2 , typename R > template<typename Partial_Function > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

If `pfunc` maps only some of the dimensions of `*this` then the rest will be projected away.

If the highest dimension mapped to by `pfunc` is higher than the highest dimension in `*this` then the number of dimensions in `this` will be increased to the highest dimension mapped to by `pfunc`.

Parameters

pfunc The partial function specifying the destiny of each space dimension.

The template class `Partial_Function` must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty codomain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the codomain of the partial function. The `max_in_codomain()` method is called at most once.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned. This method is called at most n times, where n is the dimension of the vector space enclosing `*this`.

The result is undefined if `pfunc` does not encode a partial function with the properties described in specification of the mapping operator.

Definition at line 620 of file Partially_Reduced_Product.inlines.hh.

11.151.4.47 template<typename D1 , typename D2 , typename R > dimension_type Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::max_space_dimension () [inline, static]

Returns the maximum space dimension this product can handle.

Definition at line 37 of file Partially_Reduced_Product.inlines.hh.

11.151.4.48 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *point*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if the supremum value can be reached in `this`.

point When maximization succeeds, will be assigned a generator point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded by `*this`, `false` is returned and `sup_n`, `sup_d`, `maximum` and `point` are left untouched.

Definition at line 302 of file Partially_Reduced_Product.templates.hh.

References PPL_DIRTY_TEMP_COEFFICIENT.

11.151.4.49 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if the supremum value can be reached in `this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded by `*this`, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 199 of file Partially_Reduced_Product.templates.hh.

References PPL_DIRTY_TEMP_COEFFICIENT.

11.151.4.50 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *point*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if the infimum value can be reached in `this`.

point When minimization succeeds, will be assigned a generator point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `point` are left untouched.

Definition at line 361 of file Partially_Reduced_Product.templates.hh.

References PPL_DIRTY_TEMP_COEFFICIENT.

11.151.4.51 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below i `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if the infimum value can be reached in `this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 250 of file Partially_Reduced_Product.templates.hh.

References PPL_DIRTY_TEMP_COEFFICIENT.

11.151.4.52 template<typename D1 , typename D2 , typename R > Congruence_System Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences () const [inline]

Returns a system of congruences which approximates `*this`, in reduced form.

Definition at line 81 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Congruence_System::insert(), Parma_Polyhedra_Library::Grid::minimized_congruences(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.53 template<typename D1 , typename D2 , typename R > Constraint_System Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints () const [inline]

Returns a system of constraints which approximates `*this`, in reduced form.

Definition at line 50 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Polyhedron::minimized_constraints(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.54 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 419 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

11.151.4.55 template<typename D1 , typename D2 , typename R > Partially_Reduced_Product< D1, D2, R > & Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::operator= (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

The assignment operator. (`*this` and `y` can be dimension-incompatible.).

Definition at line 471 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

11.151.4.56 template<typename D1, typename D2, typename R> void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.151.4.57 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce () const [inline]

Reduce.

Definition at line 663 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_reduced(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::set_reduced_flag().

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constrains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::difference_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain1(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain2(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_bounded(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_discrete(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_disjoint_from(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_empty(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_topologically_closed(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::strictly_contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::time_elapse_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::unconstrain(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::widening_assign().

11.151.4.58 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_congruence (const Congruence & *cg*) [inline]

Use the congruence cg to refine *this.

Parameters

cg The congruence to be used for refinement.

Exceptions

std::invalid_argument Thrown if *this and cg are dimension-incompatible.

Definition at line 405 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.59 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Use the congruences in cgs to refine *this.

Parameters

cgs The congruences to be used for refinement.

Exceptions

std::invalid_argument Thrown if *this and cgs are dimension-incompatible.

Definition at line 441 of file Partially_Reduced_Product.inlines.hh.

11.151.4.60 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_constraint (const Constraint & *c*) [inline]

Use the constraint c to refine *this.

Parameters

c The constraint to be used for refinement.

Exceptions

std::invalid_argument Thrown if *this and c are dimension-incompatible.

Definition at line 389 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::clear_reduced_flag(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.61 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_constraints (const Constraint_System & *cs*) [inline]

Use the constraints in `cs` to refine `*this`.

Parameters

cs The constraints to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 423 of file Partially_Reduced_Product.inlines.hh.

11.151.4.62 template<typename D1 , typename D2 , typename R > Poly_Gen_Relation Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with (const Generator & *g*) const [inline]

Returns the relations holding between `*this` and `g`.

Definition at line 135 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), and Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes().

11.151.4.63 template<typename D1 , typename D2 , typename R > Poly_Con_Relation Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with (const Congruence & *cg*) const [inline]

Returns the relations holding between `*this` and `cg`.

Definition at line 173 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Poly_Con_Relation::nothing(), and Parma_Polyhedra_Library::Poly_Con_Relation::saturates().

11.151.4.64 template<typename D1 , typename D2 , typename R > Poly_Con_Relation Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with (const Constraint & *c*) const [inline]

Returns the relations holding between `*this` and `c`.

Definition at line 147 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_-Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_-Polyhedra_Library::Poly_Con_Relation::nothing(), and Parma_Polyhedra_Library::Poly_Con_-Relation::saturates().

11.151.4.65 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::remove_higher_space_dimensions (dimension_type *new_dimension*) [inline]

Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_-dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimensions` is greater than the space dimension of `*this`.

Definition at line 611 of file Partially_Reduced_Product.inlines.hh.

11.151.4.66 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::remove_space_dimensions (const Variables_Set & *vars*) [inline]

Removes all the specified dimensions from the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 603 of file Partially_Reduced_Product.inlines.hh.

11.151.4.67 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::set_reduced_flag () const [inline, protected]

Sets the reduced flag.

Definition at line 688 of file Partially_Reduced_Product.inlines.hh.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.68 template<typename D1 , typename D2 , typename R > dimension_type Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 182 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::hash_code().

11.151.4.69 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::strictly_contains (const Partially_Reduced_Product< D1, D2, R > & *y*) const [inline]

Returns `true` if and only if each component of `*this` strictly contains the corresponding component of `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 654 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.70 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap (Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Swaps `*this` with product `y`. (`*this` and `y` can be dimension-incompatible.).

Definition at line 373 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap().

11.151.4.71 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::time_elapse_assign (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`. (See also time-elapse.).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 356 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.72 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 366 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2.

11.151.4.73 template<typename D1 , typename D2 , typename R > memory_size_type Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 176 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::external_memory_in_bytes().

11.151.4.74 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::unconstrain (const Variables_Set & *vars*) [inline]

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 207 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.75 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::unconstrain (Variable *var*) [inline]

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 199 of file Partially_Reduced_Product.inlines.hh.

11.151.4.76 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to `*this` an upper bound of `*this` and `y` computed on the corresponding components.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 236 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.4.77 template<typename D1 , typename D2 , typename R > bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign_if_exact (const Partially_Reduced_Product< D1, D2, R > & *y*) [inline]

Assigns to `*this` an upper bound of `*this` and `y` computed on the corresponding components. If it is exact on each of the components of `*this`, `true` is returned, otherwise `false` is returned.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 246 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce(), and Parma_Polyhedra_Library::swap().

11.151.4.78 template<typename D1 , typename D2 , typename R > void Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::widening_assign (const Partially_Reduced_Product< D1, D2, R > & *y*, unsigned * *tp* = NULL) [inline]

Assigns to `*this` the result of computing the "widening" between `*this` and `y`.

This widening uses either the congruence or generator systems depending on which of the systems describing x and y are up to date and minimized.

Parameters

y A product that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 561 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2, and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce().

11.151.5 Friends And Related Function Documentation

11.151.5.1 template<typename D1 , typename D2 , typename R > bool operator!= (const Partially_Reduced_Product< D1, D2, R > & *x*, const Partially_Reduced_Product< D1, D2, R > & *y*) [related]

Returns `true` if and only if the components of `x` and `y` are not pairwise equal.

Note that `x` and `y` may be dimension-incompatible: in those cases, the value `true` is returned.

Definition at line 726 of file Partially_Reduced_Product.inlines.hh.

11.151.5.2 template<typename D1 , typename D2 , typename R > std::ostream & operator<< (std::ostream & *s*, const Partially_Reduced_Product< D1, D2, R > & *pd*) [related]

Output operator.

Writes a textual representation of `dp` on `s`.

Definition at line 734 of file Partially_Reduced_Product.inlines.hh.

11.151.5.3 template<typename D1, typename D2, typename R> bool operator== (const Partially_Reduced_Product< D1, D2, R > & *x*, const Partially_Reduced_Product< D1, D2, R > & *y*) [friend]

Returns `true` if and only if the components of `x` and `y` are pairwise equal.

Note that `x` and `y` may be dimension-incompatible: in those cases, the value `false` is returned.

Definition at line 716 of file Partially_Reduced_Product.inlines.hh.

11.151.5.4 template<typename D1, typename D2, typename R> std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Partially_Reduced_Product< D1, D2, R > & *dp*) [friend]

11.151.5.5 template<typename D1 , typename D2 , typename R > void swap (Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > & *x*, Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R > & *y*) [related]

Specializes `std::swap`.

Definition at line 804 of file Partially_Reduced_Product.inlines.hh.

References Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap().

11.151.6 Member Data Documentation

11.151.6.1 template<typename D1, typename D2, typename R> D1 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d1 [protected]

The first component.

Definition at line 1623 of file Partially_Reduced_Product.defs.hh.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_congruence(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_constraint(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_load(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::concatenate_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constrains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::difference_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain1(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::external_memory_in_bytes(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::intersection_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_bounded(), Parma_Polyhedra_Library::Partially_

Reduced_Product< D1, D2, R >::is_discrete(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_disjoint_from(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_empty(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_topologically_closed(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_universe(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::operator=(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_congruence(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_constraint(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::space_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::strictly_contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::time_elapse_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::topological_closure_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::unconstrain(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::widening_assign().

11.151.6.2 template<typename D1, typename D2, typename R> D2 Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::d2 [protected]

The second component.

Definition at line 1626 of file Partially_Reduced_Product.defs.hh.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_congruence(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::add_constraint(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::affine_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_load(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::concatenate_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constrains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::difference_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::domain2(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::external_memory_in_bytes(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::intersection_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_bounded(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_discrete(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_disjoint_from(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_empty(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_topologically_closed(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_universe(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::operator=(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduce(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_congruence(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::refine_with_constraint(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1,

D2, R >::space_dimension(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::strictly_contains(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::time_elapse_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::topological_closure_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::unconstrain(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::upper_bound_assign_if_exact(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::widening_assign().

11.151.6.3 template<typename D1, typename D2, typename R> bool Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::reduced [protected]

Flag to record whether the components are reduced with respect to each other and the reduction class.

Definition at line 1642 of file Partially_Reduced_Product.defs.hh.

Referenced by Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::ascii_load(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::is_reduced(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::OK(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::operator=(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::Partially_Reduced_Product(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::swap().

The documentation for this class was generated from the following files:

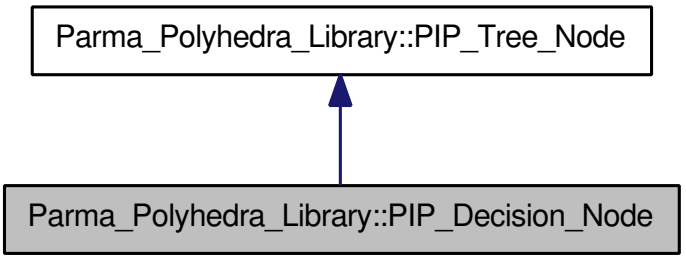
- Partially_Reduced_Product.defs.hh
- Partially_Reduced_Product.inlines.hh
- Partially_Reduced_Product.templates.hh

11.152 Parma_Polyhedra_Library::PIP_Decision_Node Class Reference

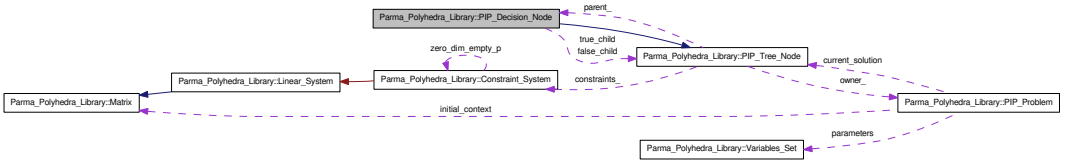
A tree node representing a decision in the space of solutions.

`#include <PIP_Tree.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::PIP_Decision_Node:



Collaboration diagram for Parma_Polyhedra_Library::PIP_Decision_Node:



Public Member Functions

- virtual PIP_Tree_Node * clone () const

  Returns a pointer to a dynamically-allocated copy of `*this`.

- virtual ~PIP_Decision_Node ()

  Destructor.

- virtual bool OK () const

  Returns `true` *if and only if* `*this` *is well formed.*

- virtual const PIP_Decision_Node * as_decision () const

  Returns `this`.

- const PIP_Tree_Node * child_node (bool b) const

  Returns a const pointer to the `b` *(true or false) branch of* `*this`.

- PIP_Tree_Node * child_node (bool b)

  Returns a pointer to the `b` *(true or false) branch of* `*this`.

- void ascii_dump (std::ostream &s) const

  Dumps to `s` *an ASCII representation of* `*this`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- virtual memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- virtual memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

Protected Member Functions

- PIP_Decision_Node (const PIP_Decision_Node &y)

  Copy constructor.

- virtual void update_tableau (const PIP_Problem &pip, dimension_type external_space_dim, dimension_type first_pending_constraint, const Constraint_Sequence &input_cs, const Variables_Set &parameters)

Implements pure virtual method PIP_Tree_Node::update_tableau.

- virtual PIP_Tree_Node * solve (const PIP_Problem &pip, bool check_feasible_context, const Matrix &context, const Variables_Set &params, dimension_type space_dim, unsigned indent_level)

  Implements pure virtual method PIP_Tree_Node::solve.

- virtual void print_tree (std::ostream &s, unsigned indent, const std::vector< bool > &pip_dim_is_-param, dimension_type first_art_dim) const

  Prints on `s` the tree rooted in `*this`.

Private Member Functions

- PIP_Decision_Node (const PIP_Problem *owner, PIP_Tree_Node *fcp, PIP_Tree_Node *tcp)

  Builds a decision node having `fcp` and `tcp` as child.

- virtual void set_owner (const PIP_Problem *owner)

  Sets the pointer to the PIP_Problem owning object.

- virtual bool check_ownership (const PIP_Problem *owner) const

  Returns `true` if and only if all the nodes in the subtree rooted in `*this` is owned by `*pip`.

Private Attributes

- PIP_Tree_Node * false_child

  Pointer to the "false" child of `*this`.

- PIP_Tree_Node * true_child

  Pointer to the "true" child of `*this`.

Friends

- class PIP_Solution_Node
- bool PIP_Problem::ascii_load (std::istream &s)

11.152.1 Detailed Description

A tree node representing a decision in the space of solutions.

Definition at line 689 of file PIP_Tree.defs.hh.

11.152.2 Constructor & Destructor Documentation

11.152.2.1 Parma_Polyhedra_Library::PIP_Decision_Node::~PIP_Decision_Node () [virtual]

Destructor.

Definition at line 557 of file PIP_Tree.cc.

References false_child, and true_child.

11.152.2.2 Parma_Polyhedra_Library::PIP_Decision_Node::PIP_Decision_Node (const PIP_Problem * *owner*, PIP_Tree_Node * *fcp*, PIP_Tree_Node * *tcp*) [explicit, private]

Builds a decision node having `fcp` and `tcp` as child.

The decision node will encode the structure "if \c cs then \p tcp else \p fcp", where the system of constraints `cs` is initially empty.

Parameters

owner Pointer to the owning PIP_Problem object; it may be null if and only if both children are null.

fcp Pointer to "false" child; it may be null.

tcp Pointer to "true" child; it may be null.

Note

If any of `fcp` or `tcp` is not null, then `owner` is required to be not null and equal to the owner of its non-null children; otherwise the behavior is undefined.

Definition at line 527 of file PIP_Tree.cc.

References false_child, Parma_Polyhedra_Library::PIP_Tree_Node::set_parent(), and true_child.

Referenced by ascii_load(), and clone().

11.152.2.3 Parma_Polyhedra_Library::PIP_Decision_Node::PIP_Decision_Node (const PIP_Decision_Node & *y*) [protected]

Copy constructor.

Definition at line 539 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::clone(), false_child, Parma_Polyhedra_Library::PIP_Tree_Node::set_parent(), and true_child.

11.152.3 Member Function Documentation

11.152.3.1 const PIP_Decision_Node * Parma_Polyhedra_Library::PIP_Decision_Node::as_decision () const [virtual]

Returns `this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 604 of file PIP_Tree.cc.

11.152.3.2 void Parma_Polyhedra_Library::PIP_Decision_Node::ascii_dump (std::ostream & *s*) const

Dumps to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 949 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::as_decision(), Parma_Polyhedra_Library::PIP_Tree_Node::as_solution(), Parma_Polyhedra_Library::PIP_Solution_Node::ascii_dump(), false_child, and true_child.

11.152.3.3 bool Parma_Polyhedra_Library::PIP_Decision_Node::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 994 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Solution_Node::ascii_load(), ascii_load(), Parma_Polyhedra_Library::PIP_Tree_Node::ascii_load(), false_child, OK(), PIP_Decision_Node(), PIP_Solution_Node, and true_child.

Referenced by ascii_load(), and Parma_Polyhedra_Library::PIP_Problem::ascii_load().

11.152.3.4 bool Parma_Polyhedra_Library::PIP_Decision_Node::check_ownership (const PIP_Problem * *owner*) const [private, virtual]

Returns `true` if and only if all the nodes in the subtree rooted in `*this` is owned by `*pip`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 582 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::check_ownership(), false_child, Parma_Polyhedra_Library::PIP_Tree_Node::get_owner(), and true_child.

11.152.3.5 PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Decision_Node::child_node (bool *b*) [inline]

Returns a pointer to the `b` (true or false) branch of `*this`.

Definition at line 102 of file PIP_Tree.inlines.hh.

References false_child, and true_child.

11.152.3.6 const PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Decision_Node::child_node (bool *b*) const [inline]

Returns a const pointer to the `b` (true or false) branch of `*this`.

Definition at line 96 of file PIP_Tree.inlines.hh.

References false_child, and true_child.

11.152.3.7 PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Decision_Node::clone () const [virtual]

Returns a pointer to a dynamically-allocated copy of `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1227 of file PIP_Tree.cc.

References PIP_Decision_Node().

11.152.3.8 memory_size_type Parma_Polyhedra_Library::PIP_Decision_Node::external_memory_in_bytes () const [virtual]

Returns the size in bytes of the memory managed by `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2863 of file PIP_Tree.cc.

References false_child, Parma_Polyhedra_Library::PIP_Tree_Node::total_memory_in_bytes(), and true_child.

Referenced by total_memory_in_bytes().

11.152.3.9 bool Parma_Polyhedra_Library::PIP_Decision_Node::OK () const [virtual]

Returns `true` if and only if `*this` is well formed.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 765 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::PIP_Tree_Node::constraints_, Parma_Polyhedra_Library::Constraint_System::end(), false_child, Parma_Polyhedra_Library::PIP_Tree_Node::OK(), and true_child.

Referenced by ascii_load(), solve(), and update_tableau().

11.152.3.10 void Parma_Polyhedra_Library::PIP_Decision_Node::print_tree (std::ostream & *s*, unsigned *indent*, const std::vector< bool > & *pip_dim_is_param*, dimension_type *first_art_dim*) const [protected, virtual]

Prints on `s` the tree rooted in `*this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2964 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::art_parameter_count(), false_child, Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print(), Parma_Polyhedra_Library::PIP_Tree_Node::print_tree(), and true_child.

11.152.3.11 void Parma_Polyhedra_Library::PIP_Decision_Node::set_owner (const PIP_Problem * *owner*) [private, virtual]

Sets the pointer to the PIP_Problem owning object.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 568 of file PIP_Tree.cc.

References false_child, Parma_Polyhedra_Library::PIP_Tree_Node::owner_, Parma_Polyhedra_Library::PIP_Tree_Node::set_owner(), and true_child.

Referenced by Parma_Polyhedra_Library::PIP_Problem::ascii_load().

11.152.3.12 PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Decision_Node::solve (const PIP_Problem & *pip*, bool *check_feasible_context*, const Matrix & *context*, const Variables_Set & *params*, dimension_type *space_dim*, unsigned *indent_level*) [protected, virtual]

Implements pure virtual method PIP_Tree_Node::solve.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 822 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::artificial_parameters, Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::PIP_Tree_Node::constraints_, Parma_Polyhedra_Library::Constraint_System::empty(), Parma_Polyhedra_Library::Constraint_System::end(), false_child, Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::PIP_Tree_Node::OK(), Parma_Polyhedra_Library::PIP_Tree_Node::parent(), Parma_Polyhedra_Library::PIP_Tree_Node::parent_merge(), Parma_Polyhedra_Library::PIP_Tree_Node::set_parent(), Parma_Polyhedra_Library::PIP_Tree_Node::solve(), Parma_Polyhedra_Library::Constraint_System::swap(), and true_child.

11.152.3.13 memory_size_type Parma_Polyhedra_Library::PIP_Decision_Node::total_memory_in_bytes () const [virtual]

Returns the total size in bytes of the memory occupied by `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2873 of file PIP_Tree.cc.

References external_memory_in_bytes().

11.152.3.14 void Parma_Polyhedra_Library::PIP_Decision_Node::update_tableau (const PIP_Problem & *pip*, dimension_type *external_space_dim*, dimension_type *first_pending_constraint*, const Constraint_Sequence & *input_cs*, const Variables_Set & *parameters*) [protected, virtual]

Implements pure virtual method PIP_Tree_Node::update_tableau.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 802 of file PIP_Tree.cc.

References false_child, OK(), true_child, and Parma_Polyhedra_Library::PIP_Tree_Node::update_tableau().

11.152.4 Friends And Related Function Documentation

11.152.4.1 bool PIP_Problem::ascii_load (std::istream & *s*) [friend]

11.152.4.2 friend class PIP_Solution_Node [friend]

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 726 of file PIP_Tree.defs.hh.

Referenced by ascii_load().

11.152.5 Member Data Documentation

11.152.5.1 PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Decision_Node::false_child [private]

Pointer to the "false" child of `*this`.

Definition at line 732 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), check_ownership(), child_node(), external_memory_in_bytes(), OK(), PIP_Decision_Node(), print_tree(), set_owner(), Parma_Polyhedra_Library::PIP_-

Solution_Node::solve(), solve(), update_tableau(), and ~PIP_Decision_Node().

11.152.5.2 PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Decision_Node::true_child [private]

Pointer to the "true" child of `*this`.

Definition at line 735 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), check_ownership(), child_node(), external_memory_in_bytes(), OK(), PIP_Decision_Node(), print_tree(), set_owner(), solve(), update_tableau(), and ~PIP_Decision_Node().

The documentation for this class was generated from the following files:

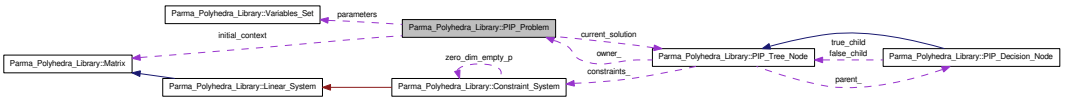
- PIP_Tree.defs.hh
- PIP_Tree.inlines.hh
- PIP_Tree.cc

11.153 Parma_Polyhedra_Library::PIP_Problem Class Reference

A Parametric Integer (linear) Programming problem.

`#include <PIP_Problem.defs.hh>`

Collaboration diagram for Parma_Polyhedra_Library::PIP_Problem:



Public Types

- enum Control_Parameter_Name { CUTTING_STRATEGY, PIVOT_ROW_STRATEGY, CONTROL_PARAMETER_NAME_SIZE }

  Possible names for PIP_Problem control parameters.

- enum Control_Parameter_Value {

  CUTTING_STRATEGY_FIRST, CUTTING_STRATEGY_DEEPEST, CUTTING_STRATEGY_ALL, PIVOT_ROW_STRATEGY_FIRST,

  PIVOT_ROW_STRATEGY_MAX_COLUMN, CONTROL_PARAMETER_VALUE_SIZE }

  Possible values for PIP_Problem control parameters.

- typedef Constraint_Sequence::const_iterator const_iterator

  A type alias for the read-only iterator on the constraints defining the feasible region.

Public Member Functions

- PIP_Problem (dimension_type dim=0)

  Builds a trivial PIP problem.

- template<typename In >
  PIP_Problem (dimension_type dim, In first, In last, const Variables_Set &p_vars)

  Builds a PIP problem having space dimension `dim` *from the sequence of constraints in the range* $[\mathrm{first}, \mathrm{last})$*; those dimensions whose indices occur in* `p_vars` *are interpreted as parameters.*

- PIP_Problem (const PIP_Problem &y)

  Ordinary copy-constructor.

- ~PIP_Problem ()

  Destructor.

- PIP_Problem & operator= (const PIP_Problem &y)

  Assignment operator.

- dimension_type space_dimension () const

  Returns the space dimension of the PIP problem.

- const Variables_Set & parameter_space_dimensions () const

  Returns a set containing all the variables' indexes representing the parameters of the PIP problem.

- const_iterator constraints_begin () const

  Returns a read-only iterator to the first constraint defining the feasible region.

- const_iterator constraints_end () const

  Returns a past-the-end read-only iterator to the sequence of constraints defining the feasible region.

- void clear ()

  Resets `*this` *to be equal to the trivial PIP problem.*

- void add_space_dimensions_and_embed (dimension_type m_vars, dimension_type m_params)

  Adds `m_vars + m_params` *new space dimensions and embeds the old PIP problem in the new vector space.*

- void add_to_parameter_space_dimensions (const Variables_Set &p_vars)

  Sets the space dimensions whose indexes which are in set `p_vars` *to be parameter space dimensions.*

- void add_constraint (const Constraint &c)

  Adds a copy of constraint `c` *to the PIP problem.*

- void add_constraints (const Constraint_System &cs)

  Adds a copy of the constraints in `cs` *to the PIP problem.*

- bool is_satisfiable () const

  Checks satisfiability of `*this`*.*

- PIP_Problem_Status solve () const

  Optimizes the PIP problem.

- PIP_Tree solution () const

  Returns a feasible solution for `*this`*, if it exists.*

- PIP_Tree optimizing_solution () const

  Returns an optimizing solution for `*this`*, if it exists.*

- bool OK () const

  Checks if all the invariants are satisfied.

- void print_solution (std::ostream &s, unsigned indent=0) const

  Prints on `s` *the solution computed for* `*this`*.*

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`*.*

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`*.*

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`*.*

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`*.*

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`*.*

- void swap (PIP_Problem &y)

  Swaps `*this` *with* `y`*.*

- Control_Parameter_Value get_control_parameter (Control_Parameter_Name name) const

  Returns the value of control parameter `name`*.*

- void set_control_parameter (Control_Parameter_Value value)

  Sets control parameter `value`*.*

- void set_big_parameter_dimension (dimension_type big_dim)

  Sets the dimension for the big parameter to `big_dim`*.*

- dimension_type get_big_parameter_dimension () const

  Returns the space dimension for the big parameter.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a PIP_Problem can handle.

Private Types

- enum Status { UNSATISFIABLE, OPTIMIZED, PARTIALLY_SATISFIABLE }

  An enumerated type describing the internal status of the PIP problem.

- typedef std::vector< Constraint > Constraint_Sequence

  A type alias for a sequence of constraints.

Private Member Functions

- void control_parameters_init ()

  Initializes the control parameters with default values.

- void control_parameters_copy (const PIP_Problem &y)

  Copies the control parameters from problem object `y`.

Private Attributes

- dimension_type external_space_dim

  The dimension of the vector space.

- dimension_type internal_space_dim

  The space dimension of the current (partial) solution of the PIP problem; it may be smaller than `external_space_dim`.

- Status status

  The internal state of the MIP problem.

- PIP_Tree_Node * current_solution

  The current solution decision tree.

- Constraint_Sequence input_cs

  The sequence of constraints describing the feasible region.

- dimension_type first_pending_constraint

  The first index of 'input_cs' containing a pending constraint.

- Variables_Set parameters

  A set containing all the indices of space dimensions that are interpreted as problem parameters.

- Matrix initial_context

The initial context.

- Control_Parameter_Value control_parameters [CONTROL_PARAMETER_NAME_SIZE]

  The control parameters for the problem object.

- dimension_type big_parameter_dimension

  The dimension for the big parameter, or `not_a_dimension()` *if not set.*

Friends

- class PIP_Solution_Node

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const PIP_Problem &p)

  Output operator.

- void swap (Parma_Polyhedra_Library::PIP_Problem &x, Parma_Polyhedra_Library::PIP_Problem &y)

  Specializes `std::swap`.

11.153.1 Detailed Description

A Parametric Integer (linear) Programming problem. An object of this class encodes a parametric integer (linear) programming problem. The PIP problem is specified by providing:

- the dimension of the vector space;
- the subset of those dimensions of the vector space that are interpreted as integer parameters (the other space dimensions are interpreted as non-parameter integer variables);
- a finite set of linear equality and (strict or non-strict) inequality constraints involving variables and/or parameters; these constraints are used to define:
  - the *feasible region*, if they involve one or more problem variable (and maybe some parameters);
  - the *initial context*, if they only involve the parameters;
- optionally, the so-called *big parameter*, i.e., a problem parameter to be considered arbitrarily big.

Note that all problem variables and problem parameters are assumed to take non-negative integer values, so that there is no need to specify non-negativity constraints.

The class provides support for the (incremental) solution of the PIP problem based on variations of the revised simplex method and on Gomory cut generation techniques.

The solution for a PIP problem is the lexicographic minimum of the integer points of the feasible region, expressed in terms of the parameters. As the problem to be solved only involves non-negative variables and parameters, the problem will always be either unfeasible or optimizable.

As the feasibility and the solution value of a PIP problem depend on the values of the parameters, the solution is a binary decision tree, dividing the context parameter set into subsets. The tree nodes are of two kinds:

- *Decision* nodes. These are internal tree nodes encoding one or more linear tests on the parameters; if all the tests are satisfied, then the solution is the node's *true* child; otherwise, the solution is the node's *false* child;
- *Solution* nodes. These are leaf nodes in the tree, encoding the solution of the problem in the current context subset, where each variable is defined in terms of a linear expression of the parameters. Solution nodes also optionally embed a set of parameter constraints: if all these constraints are satisfied, the solution is described by the node, otherwise the problem has no solution.

It may happen that a decision node has no *false* child. This means that there is no solution if at least one of the corresponding constraints is not satisfied. Decision nodes having two or more linear tests on the parameters cannot have a *false* child. Decision nodes always have a *true* child.

Both kinds of tree nodes may also contain the definition of extra parameters which are artificially introduced by the solver to enforce an integral solution. Such artificial parameters are defined by the integer division of a linear expression on the parameters by an integer coefficient.

By exploiting the incremental nature of the solver, it is possible to reuse part of the computational work already done when solving variants of a given PIP_Problem: currently, incremental resolution supports the addition of space dimensions, the addition of parameters and the addition of constraints.

Example problem

An example PIP problem can be defined the following:

```
3*j >= -2*i+8
j <= 4*i - 4
i <= n
j <= m
```

where `i` and `j` are the problem variables and `n` and `m` are the problem parameters. This problem can be optimized; the resulting solution tree may be represented as follows:

```
if 7*n >= 10 then
  if 7*m >= 12 then
    {i = 2 ; j = 2}
  else
    Parameter P = (m) div 2
    if 2*n + 3*m >= 8 then
      {i = -m - P + 4 ; j = m}
    else
      _|_
else
  _|_
```

The solution tree starts with a decision node depending on the context constraint `7*n >= 10`. If this constraint is satisfied by the values assigned to the problem parameters, then the (textually first) `then` branch is taken, reaching the *true* child of the root node (which in this case is another decision node); otherwise, the (textually last) `else` branch is taken, for which there is no corresponding *false* child.

The $\bot$ notation, also called *bottom*, denotes the lexicographic minimum of an empty set of solutions, here meaning the corresponding subproblem is unfeasible.

Notice that a tree node may introduce new (non-problem) parameters, as is the case for parameter `P` in the (textually first) `else` branch above. These *artificial* parameters are only meaningful inside the

subtree where they are defined and are used to define the parametric values of the problem variables in solution nodes (e.g., the `{i, j}` vector in the textually third `then` branch).

Context restriction

The above solution is correct in an unrestricted initial context, meaning all possible values are allowed for the parameters. If we restrict the context with the following parameter inequalities:

```
m >= n
n >= 5
```

then the resulting optimizing tree will be a simple solution node:

```
{i = 2 ; j = 2}
```

Creating the PIP_Problem object

The PIP_Problem object corresponding to the above example can be created as follows:

```
Variable i(0);
Variable j(1);
Variable n(2);
Variable m(3);
Variables_Set params(n, m);
Constraint_System cs;
cs.insert(3*j >= -2*i+8);
cs.insert(j <= 4*i - 4);
cs.insert(j <= m);
cs.insert(i <= n);
PIP_Problem pip(cs.space_dimension(), cs.begin(), cs.end(), params);
```

If you want to restrict the initial context, simply add the parameter constraints the same way as for normal constraints.

```
cs.insert(m >= n);
cs.insert(n >= 5);
```

Solving the problem

Once the PIP_Problem object has been created, you can start the resolution of the problem by calling the solve() method:

```
PIP_Problem_Status status = pip.solve();
```

where the returned `status` indicates if the problem has been optimized or if it is unfeasible for any possible configuration of the parameter values. The resolution process is also started if an attempt is made to get its solution, as follows:

```
const PIP_Tree_Node* node = pip.solution();
```

In this case, an unfeasible problem will result in an empty solution tree, i.e., assigning a null pointer to `node`.

Printing the solution tree

A previously computed solution tree may be printed as follows:

```
pip.print_solution(std::cout);
```

This will produce the following output (note: variables and parameters are printed according to the default output function; see `Variable::set_output_function`):

```
if 7*C >= 10 then
  if 7*D >= 12 then
    {2 ; 2}
  else
    Parameter E = (D) div 2
```

```
    if 2*C + 3*D >= 8 then
      {-D - E + 4 ; D}
    else
      _|_
else
  _|_
```

Spanning the solution tree

A parameter assignment for a PIP problem binds each of the problem parameters to a non-negative integer value. After fixing a parameter assignment, the "spanning" of the PIP problem solution tree refers to the process whereby the solution tree is navigated, starting from the root node: the value of artificial parameters is computed according to the parameter assignment and the node's contraints are evaluated, thereby descending in either the true or the false subtree of decision nodes and eventually reaching a solution node or a bottom node. If a solution node is found, each of the problem variables is provided with a parametric expression, which can be evaluated to a fixed value using the given parameter assignment and the computed values for artificial parameters.

The coding of the spanning process can be done as follows. First, the root of the PIP solution tree is retrieved:

```
const PIP_Tree_Node* node = pip.solution();
```

If `node` represents an unfeasible solution (i.e., $\bot$), its value will be `0`. For a non-null tree node, the virtual methods `PIP_Tree_Node::as_decision()` and `PIP_Tree_Node::as_solution()` can be used to check whether the node is a decision or a solution node:

```
const PIP_Solution_Node* sol = node->as_solution();
if (sol != 0) {
  // The node is a solution node
  ...
}
else {
  // The node is a decision node
  const PIP_Decision_Node* dec = node->as_decision();
  ...
}
```

The true (resp., false) child node of a Decision Node may be accessed by using method `PIP_Decision_Node::child_node(bool)`, passing `true` (resp., `false`) as the input argument.

Artificial parameters

A PIP_Tree_Node::Artificial_Parameter object represents the result of the integer division of a Linear_Expression (on the other parameters, including the previously-defined artificials) by an integer denominator (a Coefficient object). The dimensions of the artificial parameters (if any) in a tree node have consecutive indices starting from `dim+1`, where the value of `dim` is computed as follows:

- for the tree root node, `dim` is the space dimension of the PIP_Problem;
- for any other node of the tree, it is recusrively obtained by adding the value of `dim` computed for the parent node to the number of artificial parameters defined in the parent node.

Since the numbering of dimensions for artificial parameters follows the rule above, the addition of new problem variables and/or new problem parameters to an already solved PIP_Problem object (as done when incrementally solving a problem) will result in the systematic renumbering of all the existing artificial parameters.

Node constraints

All kind of tree nodes can contain context constraints. Decision nodes always contain at least one of them. The node's local constraint system can be obtained using method PIP_Tree_Node::constraints. These constraints only involve parameters, including both the problem parameters and the artificial parameters that have been defined in nodes occurring on the path from the root node to the current node. The meaning of these constraints is as follows:

- On a decision node, if all tests in the constraints are true, then the solution is the *true* child; otherwise it is the *false* child.
- On a solution node, if the (possibly empty) system of constraints evaluates to true for a given parameter assignment, then the solution is described by the node; otherwise the solution is $\bot$ (i.e., the problem is unfeasible for that parameter assignment).

Getting the optimal values for the variables

After spanning the solution tree using the given parameter assignment, if a solution node has been reached, then it is possible to retrieve the parametric expression for each of the problem variables using method PIP_Solution_Node::parametric_values. The retrieved expression will be defined in terms of all the parameters (problem parameters and artificial parameters defined along the path).

Solving maximization problems

You can solve a lexicographic maximization problem by reformulating its constraints using variable substitution. Proceed the following steps:

- Create a big parameter (see PIP_Problem::set_big_parameter_dimension), which we will call M.
- Reformulate each of the maximization problem constraints by substituting each x_i variable with an expression of the form $M - x'_i$, where the x'_i variables are positive variables to be minimized.
- Solve the lexicographic minimum for the x' variable vector.
- In the solution expressions, the values of the x' variables will be expressed in the form: $x'_i = M - x_i$. To get back the value of the expression of each x_i variable, just apply the formula: $x_i = M - x'_i$.

Note that if the resulting expression of one of the x'_i variables is not in the $x'_i = M - x_i$ form, this means that the sign-unrestricted problem is unbounded.

You can choose to maximize only a subset of the variables while minimizing the other variables. In that case, just apply the variable substitution method on the variables you want to be maximized. The variable optimization priority will still be in lexicographic order.

Example: consider you want to find the lexicographic maximum of the (x, y) vector, under the constraints:

$$\begin{cases} y \geq 2x - 4 \\ y \leq -x + p \end{cases}$$

where p is a parameter.

After variable substitution, the constraints become:

$$\begin{cases} M - y \geq 2M - 2x - 4 \\ M - y \leq -M + x + p \end{cases}$$

The code for creating the corresponding problem object is the following:

```
Variable x(0);
Variable y(1);
Variable p(2);
Variable M(3);
Variables_Set params(p, M);
Constraint_System cs;
cs.insert(M - y >= 2*M - 2*x - 4);
cs.insert(M - y <= -M + x + p);
PIP_Problem pip(cs.space_dimension(), cs.begin(), cs.end(), params);
pip.set_big_parameter_dimension(3);     // M is the big parameter
```

Solving the problem provides the following solution:

```
Parameter E = (C + 1) div 3
{D - E - 1 ; -C + D + E + 1}
```

Under the notations above, the solution is:

$$\begin{cases} x' = M - \left\lfloor \frac{p+1}{3} \right\rfloor - 1 \\ y' = M - p + \left\lfloor \frac{p+1}{3} \right\rfloor + 1 \end{cases}$$

Performing substitution again provides us with the values of the original variables:

$$\begin{cases} x = \left\lfloor \frac{p+1}{3} \right\rfloor + 1 \\ y = p - \left\lfloor \frac{p+1}{3} \right\rfloor - 1 \end{cases}$$

Allowing variables to be arbitrarily signed

You can deal with arbitrarily signed variables by reformulating the constraints using variable substitution. Proceed the following steps:

- Create a big parameter (see PIP_Problem::set_big_parameter_dimension), which we will call M.
- Reformulate each of the maximization problem constraints by substituting each x_i variable with an expression of the form $x'_i - M$, where the x'_i variables are positive.
- Solve the lexicographic minimum for the x' variable vector.
- The solution expression can be read in the form:
- In the solution expressions, the values of the x' variables will be expressed in the form: $x'_i = x_i + M$. To get back the value of the expression of each signed x_i variable, just apply the formula: $x_i = x'_i - M$.

Note that if the resulting expression of one of the x'_i variables is not in the $x'_i = x_i + M$ form, this means that the sign-unrestricted problem is unbounded.

You can choose to define only a subset of the variables to be sign-unrestricted. In that case, just apply the variable substitution method on the variables you want to be sign-unrestricted.

Example: consider you want to find the lexicographic minimum of the (x, y) vector, where the x and y variables are sign-unrestricted, under the constraints:

$$\begin{cases} y \geq -2x - 4 \\ 2y \leq x + 2p \end{cases}$$

where p is a parameter.

After variable substitution, the constraints become:

$$\begin{cases} y' - M \geq -2x' + 2M - 4 \\ 2y' - 2M \leq x' - M + 2p \end{cases}$$

The code for creating the corresponding problem object is the following:

```
Variable x(0);
Variable y(1);
Variable p(2);
Variable M(3);
Variables_Set params(p, M);
Constraint_System cs;
cs.insert(y - M >= -2*x + 2*M - 4);
cs.insert(2*y - 2*M <= x - M + 2*p);
PIP_Problem pip(cs.space_dimension(), cs.begin(), cs.end(), params);
pip.set_big_parameter_dimension(3);     // M is the big parameter
```

Solving the problem provides the following solution:

```
Parameter E = (2*C + 3) div 5
{D - E - 1 ; D + 2*E - 2}
```

Under the notations above, the solution is:

$$\begin{cases} x' = M - \left\lfloor \frac{2p+3}{5} \right\rfloor - 1 \\ y' = M + 2\left\lfloor \frac{2p+3}{5} \right\rfloor - 2 \end{cases}$$

Performing substitution again provides us with the values of the original variables:

$$\begin{cases} x = -\left\lfloor \frac{2p+3}{5} \right\rfloor - 1 \\ y = 2\left\lfloor \frac{2p+3}{5} \right\rfloor - 2 \end{cases}$$

Allowing parameters to be arbitrarily signed

You can consider a parameter p arbitrarily signed by replacing p with $p^+ - p^-$, where both p^+ and p^- are positive parameters. To represent a set of arbitrarily signed parameters, replace each parameter p_i with $p_i^+ - p^-$, where $-p^-$ is the minimum negative value of all parameters.

Minimizing a linear cost function

Lexicographic solving can be used to find the parametric minimum of a linear cost function.

Suppose the variables are named $x_1, x_2, \ldots, x_n$, and the parameters $p_1, p_2, \ldots, p_m$. You can minimize a linear cost function $f(x_2, \ldots, x_n, p_1, \ldots, p_m)$ by simply adding the constraint $x_1 \geq f(x_2, \ldots, x_n, p_1, \ldots, p_m)$ to the constraint system. As lexicographic minimization ensures x_1 is minimized in priority, and because x_1 is forced by a constraint to be superior or equal to the cost function, optimal solutions of the problem necessarily ensure that the solution value of x_1 is the optimal value of the cost function.

Definition at line 488 of file PIP_Problem.defs.hh.

11.153.2 Member Typedef Documentation

11.153.2.1 typedef Constraint_Sequence::const_iterator Parma_Polyhedra_Library::PIP_Problem::const_iterator

A type alias for the read-only iterator on the constraints defining the feasible region.

Definition at line 567 of file PIP_Problem.defs.hh.

11.153.2.2 typedef std::vector<Constraint> Parma_Polyhedra_Library::PIP_Problem::Constraint_Sequence [private]

A type alias for a sequence of constraints.

Definition at line 560 of file PIP_Problem.defs.hh.

11.153.3 Member Enumeration Documentation

11.153.3.1 enum Parma_Polyhedra_Library::PIP_Problem::Control_Parameter_Name

Possible names for PIP_Problem control parameters.

Enumerator:

CUTTING_STRATEGY Cutting strategy.

PIVOT_ROW_STRATEGY Pivot row strategy.

CONTROL_PARAMETER_NAME_SIZE Number of different enumeration values.

Definition at line 703 of file PIP_Problem.defs.hh.

11.153.3.2 enum Parma_Polyhedra_Library::PIP_Problem::Control_Parameter_Value

Possible values for PIP_Problem control parameters.

Enumerator:

CUTTING_STRATEGY_FIRST Choose the first non-integer row.

CUTTING_STRATEGY_DEEPEST Choose row which generates the deepest cut.

CUTTING_STRATEGY_ALL Always generate all possible cuts.

PIVOT_ROW_STRATEGY_FIRST Choose the first row with negative parameter sign.

PIVOT_ROW_STRATEGY_MAX_COLUMN Choose the row which generates the lexico-maximal pivot column.

CONTROL_PARAMETER_VALUE_SIZE Number of different enumeration values.

Definition at line 715 of file PIP_Problem.defs.hh.

11.153.3.3 enum Parma_Polyhedra_Library::PIP_Problem::Status [private]

An enumerated type describing the internal status of the PIP problem.

Enumerator:

UNSATISFIABLE The PIP problem is unsatisfiable.

OPTIMIZED The PIP problem is optimized; the solution tree has been computed.

PARTIALLY_SATISFIABLE The feasible region of the PIP problem has been changed by adding new variables, parameters or constraints; a feasible solution for the old feasible region is still available.

Definition at line 768 of file PIP_Problem.defs.hh.

11.153.4 Constructor & Destructor Documentation

11.153.4.1 Parma_Polyhedra_Library::PIP_Problem::PIP_Problem (dimension_type *dim* = 0) [explicit]

Builds a trivial PIP problem.

A trivial PIP problem requires to compute the lexicographic minimum on a vector space under no constraints and with no parameters: due to the implicit non-negativity constraints, the origin of the vector space is an optimal solution.

Parameters

dim The dimension of the vector space enclosing `*this` (optional argument with default value 0).

Exceptions

std::length_error Thrown if `dim` exceeds `max_space_dimension()`.

Definition at line 47 of file PIP_Problem.cc.

References control_parameters_init(), max_space_dimension(), and OK().

11.153.4.2 template<typename In > Parma_Polyhedra_Library::PIP_Problem::PIP_Problem (dimension_type *dim*, In *first*, In *last*, const Variables_Set & *p_vars*) [inline]

Builds a PIP problem having space dimension `dim` from the sequence of constraints in the range $[\mathrm{first}, \mathrm{last})$; those dimensions whose indices occur in `p_vars` are interpreted as parameters.

Parameters

dim The dimension of the vector space (variables and parameters) enclosing `*this`.

first An input iterator to the start of the sequence of constraints.

last A past-the-end input iterator to the sequence of constraints.

p_vars The set of variables' indexes that are interpreted as parameters.

Exceptions

std::length_error Thrown if `dim` exceeds `max_space_dimension()`.

std::invalid_argument Thrown if the space dimension of a constraint in the sequence (resp., the parameter variables) is strictly greater than `dim`.

Definition at line 32 of file PIP_Problem.templates.hh.

References control_parameters_init(), external_space_dim, input_cs, max_space_dimension(), OK(), and Parma_Polyhedra_Library::Variables_Set::space_dimension().

11.153.4.3 Parma_Polyhedra_Library::PIP_Problem::PIP_Problem (const PIP_Problem & *y*)

Ordinary copy-constructor.

Definition at line 66 of file PIP_Problem.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::clone(), control_parameters_copy(), current_solution, OK(), and Parma_Polyhedra_Library::PIP_Tree_Node::set_owner().

11.153.4.4 Parma_Polyhedra_Library::PIP_Problem::~PIP_Problem ()

Destructor.

Definition at line 84 of file PIP_Problem.cc.

References current_solution.

11.153.5 Member Function Documentation

11.153.5.1 void Parma_Polyhedra_Library::PIP_Problem::add_constraint (const Constraint & *c*)

Adds a copy of constraint `c` to the PIP problem.

Exceptions

std::invalid_argument Thrown if the space dimension of `c` is strictly greater than the space dimension of `*this`.

Definition at line 625 of file PIP_Problem.cc.

References external_space_dim, input_cs, PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::Constraint::space_dimension(), status, and UNSATISFIABLE.

Referenced by add_constraints().

11.153.5.2 void Parma_Polyhedra_Library::PIP_Problem::add_constraints (const Constraint_System & *cs*)

Adds a copy of the constraints in `cs` to the PIP problem.

Exceptions

std::invalid_argument Thrown if the space dimension of constraint system `cs` is strictly greater than the space dimension of `*this`.

Definition at line 640 of file PIP_Problem.cc.

References add_constraint(), Parma_Polyhedra_Library::Constraint_System::begin(), and Parma_Polyhedra_Library::Constraint_System::end().

11.153.5.3 void Parma_Polyhedra_Library::PIP_Problem::add_space_dimensions_and_embed (dimension_type *m_vars*, dimension_type *m_params*)

Adds `m_vars + m_params` new space dimensions and embeds the old PIP problem in the new vector space.

Parameters

m_vars The number of space dimensions to add that are interpreted as PIP problem variables (i.e., non parameters). These are added *before* adding the `m_params` parameters.

m_params The number of space dimensions to add that are interpreted as PIP problem parameters. These are added *after* having added the `m_vars` problem variables.

Exceptions

std::length_error Thrown if adding `m_vars + m_params` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

The new space dimensions will be those having the highest indexes in the new PIP problem; they are initially unconstrained.

Definition at line 565 of file PIP_Problem.cc.

References external_space_dim, Parma_Polyhedra_Library::Variables_Set::insert(), max_space_dimension(), OK(), parameters, PARTIALLY_SATISFIABLE, space_dimension(), status, and UNSATISFIABLE.

11.153.5.4 void Parma_Polyhedra_Library::PIP_Problem::add_to_parameter_space_dimensions (const Variables_Set & *p_vars*)

Sets the space dimensions whose indexes which are in set `p_vars` to be parameter space dimensions.

Exceptions

std::invalid_argument Thrown if some index in `p_vars` does not correspond to a space dimension in `*this`.

Definition at line 600 of file PIP_Problem.cc.

References external_space_dim, Parma_Polyhedra_Library::Variables_Set::insert(), internal_space_dim, parameters, PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::Variables_Set::space_dimension(), status, and UNSATISFIABLE.

11.153.5.5 void Parma_Polyhedra_Library::PIP_Problem::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 344 of file PIP_Problem.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::as_decision(), Parma_Polyhedra_Library::PIP_Tree_Node::as_solution(), Parma_Polyhedra_Library::PIP_Solution_Node::ascii_dump(), Parma_Polyhedra_Library::Matrix::ascii_dump(), Parma_Polyhedra_Library::Variables_Set::ascii_dump(), ascii_dump(), big_parameter_dimension, CONTROL_PARAMETER_NAME_SIZE, control_parameters, current_solution, CUTTING_STRATEGY_ALL, CUTTING_STRATEGY_DEEPEST, CUTTING_STRATEGY_FIRST, external_space_dim, first_pending_constraint, initial_context, input_cs, internal_space_dim, OPTIMIZED, parameters, PARTIALLY_SATISFIABLE, PIVOT_ROW_STRATEGY_FIRST, PIVOT_ROW_STRATEGY_MAX_COLUMN, status, and UNSATISFIABLE.

11.153.5.6 void Parma_Polyhedra_Library::PIP_Problem::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by ascii_dump(), and OK().

11.153.5.7 bool Parma_Polyhedra_Library::PIP_Problem::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 422 of file PIP_Problem.cc.

References Parma_Polyhedra_Library::PIP_Solution_Node::ascii_load(), Parma_Polyhedra_Library::PIP_Decision_Node::ascii_load(), Parma_Polyhedra_Library::Constraint::ascii_load(), Parma_Polyhedra_Library::PIP_Solution_Node::set_owner(), Parma_Polyhedra_Library::PIP_Decision_Node::set_owner(), and Parma_Polyhedra_Library::Constraint::zero_dim_positivity().

11.153.5.8 void Parma_Polyhedra_Library::PIP_Problem::clear ()

Resets `*this` to be equal to the trivial PIP problem.

The space dimension is reset to 0.

Definition at line 547 of file PIP_Problem.cc.

References big_parameter_dimension, Parma_Polyhedra_Library::Matrix::clear(), control_parameters_init(), current_solution, external_space_dim, first_pending_constraint, initial_context, input_cs, internal_-

space_dim, Parma_Polyhedra_Library::not_a_dimension(), parameters, PARTIALLY_SATISFIABLE, and status.

11.153.5.9 PIP_Problem::const_iterator Parma_Polyhedra_Library::PIP_Problem::constraints_begin () const [inline]

Returns a read-only iterator to the first constraint defining the feasible region.

Definition at line 40 of file PIP_Problem.inlines.hh.

References input_cs.

Referenced by operator<<().

11.153.5.10 PIP_Problem::const_iterator Parma_Polyhedra_Library::PIP_Problem::constraints_end () const [inline]

Returns a past-the-end read-only iterator to the sequence of constraints defining the feasible region.

Definition at line 45 of file PIP_Problem.inlines.hh.

References input_cs.

Referenced by operator<<().

11.153.5.11 void Parma_Polyhedra_Library::PIP_Problem::control_parameters_copy (const PIP_Problem & *y*) [private]

Copies the control parameters from problem object y.

Definition at line 95 of file PIP_Problem.cc.

References CONTROL_PARAMETER_NAME_SIZE, and control_parameters.

Referenced by PIP_Problem().

11.153.5.12 void Parma_Polyhedra_Library::PIP_Problem::control_parameters_init () [private]

Initializes the control parameters with default values.

Definition at line 89 of file PIP_Problem.cc.

References control_parameters, CUTTING_STRATEGY, CUTTING_STRATEGY_FIRST, PIVOT_ROW_STRATEGY, and PIVOT_ROW_STRATEGY_FIRST.

Referenced by clear(), and PIP_Problem().

11.153.5.13 PPL::memory_size_type Parma_Polyhedra_Library::PIP_Problem::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Definition at line 689 of file PIP_Problem.cc.

References current_solution, Parma_Polyhedra_Library::Matrix::external_memory_in_bytes(), initial_context, input_cs, parameters, and Parma_Polyhedra_Library::PIP_Tree_Node::total_memory_in_bytes().

Referenced by total_memory_in_bytes().

11.153.5.14 dimension_type Parma_Polyhedra_Library::PIP_Problem::get_big_parameter_dimension () const [inline]

Returns the space dimension for the big parameter.

If a big parameter was not set, returns `not_a_dimension()`.

Definition at line 83 of file PIP_Problem.inlines.hh.

References big_parameter_dimension.

Referenced by operator<<().

11.153.5.15 PIP_Problem::Control_Parameter_Value Parma_Polyhedra_Library::PIP_Problem::get_control_parameter (Control_Parameter_Name *name*) const [inline]

Returns the value of control parameter `name`.

Definition at line 77 of file PIP_Problem.inlines.hh.

References CONTROL_PARAMETER_NAME_SIZE, and control_parameters.

11.153.5.16 bool Parma_Polyhedra_Library::PIP_Problem::is_satisfiable () const

Checks satisfiability of `*this`.

Returns

`true` if and only if the PIP problem is satisfiable.

Definition at line 647 of file PIP_Problem.cc.

References OPTIMIZED, PARTIALLY_SATISFIABLE, solve(), and status.

11.153.5.17 dimension_type Parma_Polyhedra_Library::PIP_Problem::max_space_dimension () [inline, static]

Returns the maximum space dimension a PIP_Problem can handle.

Definition at line 35 of file PIP_Problem.inlines.hh.

Referenced by add_space_dimensions_and_embed(), and PIP_Problem().

11.153.5.18 bool Parma_Polyhedra_Library::PIP_Problem::OK () const

Checks if all the invariants are satisfied.

Definition at line 246 of file PIP_Problem.cc.

References ascii_dump(), big_parameter_dimension, Parma_Polyhedra_Library::PIP_Tree_-Node::check_ownership(), control_parameters, current_solution, CUTTING_STRATEGY, CUTTING_-STRATEGY_ALL, CUTTING_STRATEGY_DEEPEST, CUTTING_STRATEGY_FIRST, external_-space_dim, initial_context, input_cs, internal_space_dim, Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::PIP_Tree_Node::OK(), Parma_Polyhedra_Library::Matrix::OK(), Parma_-Polyhedra_Library::Variables_Set::OK(), parameters, PIVOT_ROW_STRATEGY, PIVOT_ROW_-STRATEGY_FIRST, PIVOT_ROW_STRATEGY_MAX_COLUMN, and space_dimension().

Referenced by add_space_dimensions_and_embed(), PIP_Problem(), and solve().

11.153.5.19 PIP_Problem & Parma_Polyhedra_Library::PIP_Problem::operator= (const PIP_Problem & *y*) [inline]

Assignment operator.

Definition at line 70 of file PIP_Problem.inlines.hh.

References swap().

11.153.5.20 PPL::PIP_Tree Parma_Polyhedra_Library::PIP_Problem::optimizing_solution () const

Returns an optimizing solution for `*this`, if it exists.

A null pointer is returned for an unfeasible PIP problem.

Definition at line 239 of file PIP_Problem.cc.

References current_solution, PARTIALLY_SATISFIABLE, solve(), and status.

11.153.5.21 const Variables_Set & Parma_Polyhedra_Library::PIP_Problem::parameter_space_-dimensions () const [inline]

Returns a set containing all the variables' indexes representing the parameters of the PIP problem.

Definition at line 50 of file PIP_Problem.inlines.hh.

References parameters.

Referenced by operator<<(), Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values(), Parma_Polyhedra_Library::PIP_Tree_Node::print(), and Parma_Polyhedra_Library::PIP_Solution_-Node::update_solution().

11.153.5.22 void Parma_Polyhedra_Library::PIP_Problem::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.153.5.23 void Parma_Polyhedra_Library::PIP_Problem::print_solution (std::ostream & *s*, unsigned *indent* = 0) const

Prints on `s` the solution computed for `*this`.

Parameters

s The output stream.

indent An indentation parameter (default value 0).

Exceptions

std::logic_error Thrown if trying to print the solution when the PIP problem still has to be solved.

Definition at line 711 of file PIP_Problem.cc.

References current_solution, Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print(), OPTIMIZED, PARTIALLY_SATISFIABLE, Parma_Polyhedra_Library::PIP_Tree_Node::print(), status, and UNSATISFIABLE.

11.153.5.24 void Parma_Polyhedra_Library::PIP_Problem::set_big_parameter_dimension (dimension_type *big_dim*)

Sets the dimension for the big parameter to `big_dim`.

Definition at line 675 of file PIP_Problem.cc.

References big_parameter_dimension, internal_space_dim, and parameters.

11.153.5.25 void Parma_Polyhedra_Library::PIP_Problem::set_control_parameter (Control_Parameter_Value *value*)

Sets control parameter `value`.

Definition at line 654 of file PIP_Problem.cc.

References control_parameters, CUTTING_STRATEGY, CUTTING_STRATEGY_ALL, CUTTING_-STRATEGY_DEEPEST, CUTTING_STRATEGY_FIRST, PIVOT_ROW_STRATEGY, PIVOT_ROW_-STRATEGY_FIRST, and PIVOT_ROW_STRATEGY_MAX_COLUMN.

11.153.5.26 PPL::PIP_Tree Parma_Polyhedra_Library::PIP_Problem::solution () const

Returns a feasible solution for `*this`, if it exists.

A null pointer is returned for an unfeasible PIP problem.

Definition at line 232 of file PIP_Problem.cc.

References current_solution, PARTIALLY_SATISFIABLE, solve(), and status.

11.153.5.27 PPL::PIP_Problem_Status Parma_Polyhedra_Library::PIP_Problem::solve () const

Optimizes the PIP problem.

Returns

A PIP_Problem_Status flag indicating the outcome of the optimization attempt (unfeasible or optimized problem).

Definition at line 101 of file PIP_Problem.cc.

References Parma_Polyhedra_Library::Matrix::add_row(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), current_solution, external_space_dim, first_pending_constraint, Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), initial_context, input_cs, internal_space_dim, Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), OPTIMIZED, Parma_Polyhedra_Library::OPTIMIZED_PIP_PROBLEM, parameters, PARTIALLY_SATISFIABLE, PIP_Solution_Node, Parma_Polyhedra_Library::PIP_Tree_Node::solve(), Parma_Polyhedra_Library::Constraint::space_dimension(), status, Parma_Polyhedra_Library::UNFEASIBLE_PIP_PROBLEM, UNSATISFIABLE, and Parma_Polyhedra_Library::PIP_Tree_Node::update_tableau().

Referenced by is_satisfiable(), optimizing_solution(), and solution().

11.153.5.28 dimension_type Parma_Polyhedra_Library::PIP_Problem::space_dimension () const [inline]

Returns the space dimension of the PIP problem.

Definition at line 30 of file PIP_Problem.inlines.hh.

References external_space_dim.

Referenced by add_space_dimensions_and_embed(), OK(), operator<<(), Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values(), Parma_Polyhedra_Library::PIP_Tree_Node::print(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_solution().

11.153.5.29 void Parma_Polyhedra_Library::PIP_Problem::swap (PIP_Problem & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 55 of file PIP_Problem.inlines.hh.

References big_parameter_dimension, CONTROL_PARAMETER_NAME_SIZE, control_parameters, current_solution, external_space_dim, first_pending_constraint, initial_context, input_cs, internal_space_dim, parameters, status, and Parma_Polyhedra_Library::swap().

Referenced by operator=(), and swap().

11.153.5.30 PPL::memory_size_type Parma_Polyhedra_Library::PIP_Problem::total_memory_in_bytes () const

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 706 of file PIP_Problem.cc.

References external_memory_in_bytes().

11.153.6 Friends And Related Function Documentation

11.153.6.1 std::ostream & operator<< (std::ostream & *s*, const PIP_Problem & *pip*) [related]

Output operator.

Definition at line 32 of file PIP_Problem.cc.

References constraints_begin(), constraints_end(), get_big_parameter_dimension(), Parma_Polyhedra_Library::not_a_dimension(), parameter_space_dimensions(), and space_dimension().

11.153.6.2 friend class PIP_Solution_Node [friend]

Definition at line 816 of file PIP_Problem.defs.hh.

Referenced by solve().

11.153.6.3 void swap (Parma_Polyhedra_Library::PIP_Problem & *x*, Parma_Polyhedra_Library::PIP_Problem & *y*) [related]

Specializes `std::swap`.

Definition at line 93 of file PIP_Problem.inlines.hh.

References swap().

11.153.7 Member Data Documentation

11.153.7.1 dimension_type Parma_Polyhedra_Library::PIP_Problem::big_parameter_dimension [private]

The dimension for the big parameter, or `not_a_dimension()` if not set.

Definition at line 814 of file PIP_Problem.defs.hh.

Referenced by ascii_dump(), clear(), get_big_parameter_dimension(), OK(), set_big_parameter_dimension(), swap(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.153.7.2 Control_Parameter_Value Parma_Polyhedra_Library::PIP_Problem::control_parameters[CONTROL_PARAMETER_NAME_SIZE] [private]

The control parameters for the problem object.

Definition at line 808 of file PIP_Problem.defs.hh.

Referenced by ascii_dump(), control_parameters_copy(), control_parameters_init(), get_control_parameter(), OK(), set_control_parameter(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and swap().

11.153.7.3 PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Problem::current_solution [private]

The current solution decision tree.

Definition at line 785 of file PIP_Problem.defs.hh.

Referenced by ascii_dump(), clear(), external_memory_in_bytes(), OK(), optimizing_solution(), PIP_Problem(), print_solution(), solution(), solve(), swap(), and ~PIP_Problem().

11.153.7.4 dimension_type Parma_Polyhedra_Library::PIP_Problem::external_space_dim [private]

The dimension of the vector space.

Definition at line 759 of file PIP_Problem.defs.hh.

Referenced by add_constraint(), add_space_dimensions_and_embed(), add_to_parameter_space_dimensions(), ascii_dump(), clear(), OK(), PIP_Problem(), solve(), space_dimension(), swap(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.153.7.5 dimension_type Parma_Polyhedra_Library::PIP_Problem::first_pending_constraint [private]

The first index of 'input_cs' containing a pending constraint.

Definition at line 791 of file PIP_Problem.defs.hh.

Referenced by ascii_dump(), clear(), solve(), and swap().

11.153.7.6 Matrix Parma_Polyhedra_Library::PIP_Problem::initial_context [private]

The initial context.

Contains problem constraints on parameters only

Definition at line 804 of file PIP_Problem.defs.hh.

Referenced by ascii_dump(), clear(), external_memory_in_bytes(), OK(), solve(), and swap().

11.153.7.7 Constraint_Sequence Parma_Polyhedra_Library::PIP_Problem::input_cs [private]

The sequence of constraints describing the feasible region.

Definition at line 788 of file PIP_Problem.defs.hh.

Referenced by add_constraint(), ascii_dump(), clear(), constraints_begin(), constraints_end(), external_memory_in_bytes(), OK(), PIP_Problem(), solve(), and swap().

11.153.7.8 dimension_type Parma_Polyhedra_Library::PIP_Problem::internal_space_dim [private]

The space dimension of the current (partial) solution of the PIP problem; it may be smaller than `external_space_dim`.

Definition at line 765 of file PIP_Problem.defs.hh.

Referenced by add_to_parameter_space_dimensions(), ascii_dump(), clear(), OK(), set_big_parameter_dimension(), solve(), swap(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

11.153.7.9 Variables_Set Parma_Polyhedra_Library::PIP_Problem::parameters [private]

A set containing all the indices of space dimensions that are interpreted as problem parameters.

Definition at line 797 of file PIP_Problem.defs.hh.

Referenced by add_space_dimensions_and_embed(), add_to_parameter_space_dimensions(), ascii_dump(), clear(), external_memory_in_bytes(), OK(), parameter_space_dimensions(), set_big_parameter_dimension(), solve(), and swap().

11.153.7.10 Status Parma_Polyhedra_Library::PIP_Problem::status [private]

The internal state of the MIP problem.

Definition at line 782 of file PIP_Problem.defs.hh.

Referenced by add_constraint(), add_space_dimensions_and_embed(), add_to_parameter_space_dimensions(), ascii_dump(), clear(), is_satisfiable(), optimizing_solution(), print_solution(), solution(), solve(), and swap().

The documentation for this class was generated from the following files:

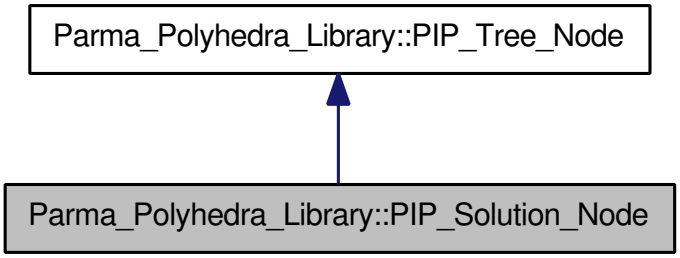
- PIP_Problem.defs.hh
- PIP_Problem.inlines.hh
- PIP_Problem.templates.hh
- PIP_Problem.cc

11.154 Parma_Polyhedra_Library::PIP_Solution_Node Class Reference

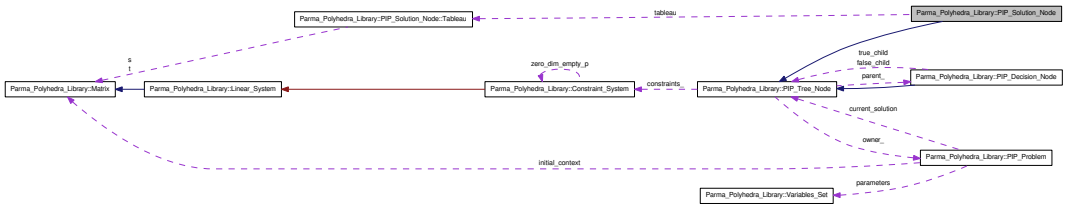
A tree node representing part of the space of solutions.

`#include <PIP_Tree.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::PIP_Solution_Node:



Collaboration diagram for Parma_Polyhedra_Library::PIP_Solution_Node:



Classes

- struct No_Constraints

  A tag type to select the alternative copy constructor.

- struct Tableau

  The type for parametric simplex tableau.

Public Member Functions

- PIP_Solution_Node (const PIP_Problem *owner)

  Constructor: builds a solution node owned by `*owner`.

- virtual PIP_Tree_Node * clone () const

  Returns a pointer to a dynamically-allocated copy of `*this`.

- virtual ~PIP_Solution_Node ()

  Destructor.

- virtual bool OK () const

  Returns `true` *if and only if* `*this` *is well formed.*

- virtual const PIP_Solution_Node * as_solution () const

  Returns `this`.

- const Linear_Expression & parametric_values (Variable var) const

  Returns a parametric expression for the values of problem variable `var`.

- void ascii_dump (std::ostream &s) const

  Dumps to `s` *an ASCII representation of* `*this`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- virtual memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- virtual memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

Protected Member Functions

- PIP_Solution_Node (const PIP_Solution_Node &y)

  Copy constructor.

- PIP_Solution_Node (const PIP_Solution_Node &y, No_Constraints)

  Alternative copy constructor.

- virtual void set_owner (const PIP_Problem *owner)

  Sets the pointer to the PIP_Problem owning object.

- virtual bool check_ownership (const PIP_Problem *owner) const

  Returns `true` *if and only if all the nodes in the subtree rooted in* `*this` *is owned by* `*pip`.

- virtual void update_tableau (const PIP_Problem &pip, dimension_type external_space_dim, dimension_type first_pending_constraint, const Constraint_Sequence &input_cs, const Variables_Set &parameters)

Implements pure virtual method PIP_Tree_Node::update_tableau.

- void update_solution (const std::vector< bool > &pip_dim_is_param) const

  Update the solution values.

- void update_solution () const

  Helper method.

- virtual PIP_Tree_Node * solve (const PIP_Problem &pip, bool check_feasible_context, const Matrix &context, const Variables_Set &params, dimension_type space_dim, unsigned indent_level)

  Implements pure virtual method PIP_Tree_Node::solve.

- void generate_cut (dimension_type i, Variables_Set &parameters, Matrix &context, dimension_type &space_dimension, unsigned indent_level)

  Generate a Gomory cut using non-integer tableau row `i`.

- virtual void print_tree (std::ostream &s, unsigned indent, const std::vector< bool > &pip_dim_is_-param, dimension_type first_art_dim) const

  Prints on `s` the tree rooted in `*this`.

Private Types

- enum Row_Sign {
  UNKNOWN, ZERO, POSITIVE, NEGATIVE,
  MIXED }

  The possible values for the sign of a parametric linear expression.

Static Private Member Functions

- static Row_Sign row_sign (const Row &x, dimension_type big_dimension)

  Returns the sign of row `x`.

Private Attributes

- Tableau tableau

  The parametric simplex tableau.

- std::vector< bool > basis

  A boolean vector for identifying the basic variables.

- std::vector< dimension_type > mapping

  A mapping between the tableau rows/columns and the original variables.

- std::vector< dimension_type > var_row

  The variable identifiers associated to the rows of the simplex tableau.

- std::vector< dimension_type > var_column

  The variable identifiers associated to the columns of the simplex tableau.

- dimension_type special_equality_row

  The variable number of the special inequality used for modelling equality constraints.

- dimension_type big_dimension

  The column index in the parametric part of the simplex tableau corresponding to the big parameter; `not_a_dimension()` *if not set.*

- std::vector< Row_Sign > sign

  A cache for computed sign values of constraint parametric RHS.

- std::vector< Linear_Expression > solution

  Parametric values for the solution.

- bool solution_valid

  An indicator for solution validity.

Friends

- bool PIP_Problem::ascii_load (std::istream &s)

11.154.1 Detailed Description

A tree node representing part of the space of solutions.

Definition at line 344 of file PIP_Tree.defs.hh.

11.154.2 Member Enumeration Documentation

11.154.2.1 enum Parma_Polyhedra_Library::PIP_Solution_Node::Row_Sign [private]

The possible values for the sign of a parametric linear expression.

Enumerator:

UNKNOWN Not computed yet (default).

ZERO All row coefficients are zero.

POSITIVE All nonzero row coefficients are positive.

NEGATIVE All nonzero row coefficients are negative.

MIXED The row contains both positive and negative coefficients.

Definition at line 573 of file PIP_Tree.defs.hh.

11.154.3 Constructor & Destructor Documentation

11.154.3.1 Parma_Polyhedra_Library::PIP_Solution_Node::PIP_Solution_Node (const PIP_Problem * *owner*) [explicit]

Constructor: builds a solution node owned by `*owner`.

Definition at line 481 of file PIP_Tree.cc.

Referenced by clone(), and solve().

11.154.3.2 Parma_Polyhedra_Library::PIP_Solution_Node::~PIP_Solution_Node () [virtual]

Destructor.

Definition at line 524 of file PIP_Tree.cc.

11.154.3.3 Parma_Polyhedra_Library::PIP_Solution_Node::PIP_Solution_Node (const PIP_Solution_Node & *y*) [protected]

Copy constructor.

Definition at line 495 of file PIP_Tree.cc.

11.154.3.4 Parma_Polyhedra_Library::PIP_Solution_Node::PIP_Solution_Node (const PIP_Solution_Node & *y*, No_Constraints) [protected]

Alternative copy constructor.

This constructor differs from the default copy constructor in that it will not copy the constraint system, nor the artificial parameters.

Definition at line 509 of file PIP_Tree.cc.

11.154.4 Member Function Documentation

11.154.4.1 const PIP_Solution_Node * Parma_Polyhedra_Library::PIP_Solution_Node::as_solution () const [virtual]

Returns `this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 599 of file PIP_Tree.cc.

11.154.4.2 void Parma_Polyhedra_Library::PIP_Solution_Node::ascii_dump (std::ostream & *s*) const

Dumps to `s` an ASCII representation of `*this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1262 of file PIP_Tree.cc.

References big_dimension, MIXED, NEGATIVE, POSITIVE, sign, solution, solution_valid, special_equality_row, UNKNOWN, var_column, var_row, and ZERO.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::ascii_dump(), and Parma_Polyhedra_Library::PIP_Problem::ascii_dump().

11.154.4.3 bool Parma_Polyhedra_Library::PIP_Solution_Node::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1331 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Linear_Expression::ascii_load(), Parma_Polyhedra_Library::PIP_Tree_Node::ascii_load(), big_dimension, MIXED, NEGATIVE, OK(), POSITIVE, sign, solution, solution_valid, special_equality_row, UNKNOWN, var_column, var_row, and ZERO.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::ascii_load(), and Parma_Polyhedra_Library::PIP_Problem::ascii_load().

11.154.4.4 bool Parma_Polyhedra_Library::PIP_Solution_Node::check_ownership (const PIP_Problem * *owner*) const [protected, virtual]

Returns `true` if and only if all the nodes in the subtree rooted in `*this` is owned by `*pip`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 577 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::get_owner().

11.154.4.5 PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Solution_Node::clone () const [virtual]

Returns a pointer to a dynamically-allocated copy of `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1222 of file PIP_Tree.cc.

References PIP_Solution_Node().

11.154.4.6 memory_size_type Parma_Polyhedra_Library::PIP_Solution_Node::external_memory_in_bytes () const [virtual]

Returns the size in bytes of the memory managed by `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2885 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Checked::bool, sign, solution, var_column, and var_row.

Referenced by total_memory_in_bytes().

11.154.4.7 void Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut (dimension_type *i*, Variables_Set & *parameters*, Matrix & *context*, dimension_type & *space_dimension*, unsigned *indent_level*) [protected]

Generate a Gomory cut using non-integer tableau row `i`.

Parameters

i Row index in simplex tableau from which the cut is generated

parameters A std::set of the current parameter dimensions (including artificials); to be updated if a new artificial parameter is to be created

context A set of linear inequalities on the parameters, in matrix form; to be updated if a new artificial parameter is to be created

space_dimension The current space dimension, including variables and all parameters; to be updated if an extra parameter is to be created

indent_level The indentation level (for debugging output only).

Definition at line 2631 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::PIP_Tree_Node::artificial_parameters, Parma_Polyhedra_Library::Variables_Set::insert(), Parma_Polyhedra_Library::neg_assign(), NEGATIVE, Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::PIP_Tree_Node::parent(), PPL_DIRTY_TEMP_COEFFICIENT, sign, var_row, WEIGHT_ADD, WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by solve().

11.154.4.8 bool Parma_Polyhedra_Library::PIP_Solution_Node::OK () const [virtual]

Returns `true` if and only if `*this` is well formed.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 697 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::OK(), var_column, and var_row.

Referenced by ascii_load(), solve(), and update_tableau().

11.154.4.9 const Linear_Expression & Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values (Variable *var*) const

Returns a parametric expression for the values of problem variable `var`.

The returned linear expression may involve problem parameters as well as artificial parameters.

Parameters

var The problem variable which is queried about.

Exceptions

std::invalid_argument Thrown if `var` is dimension-incompatible with the PIP_Problem owning this solution node, or if `var` is a problem parameter.

Definition at line 3016 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::get_owner(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::PIP_Problem::parameter_space_dimensions(), solution, Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::PIP_Problem::space_dimension(), and update_solution().

11.154.4.10 void Parma_Polyhedra_Library::PIP_Solution_Node::print_tree (std::ostream & *s*, unsigned *indent*, const std::vector< bool > & *pip_dim_is_param*, dimension_type *first_art_dim*) const [protected, virtual]

Prints on `s` the tree rooted in `*this`.

Reimplemented from Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2985 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::constraints_, Parma_Polyhedra_Library::Constraint_System::empty(), Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print(), solution, and update_solution().

11.154.4.11 PIP_Solution_Node::Row_Sign Parma_Polyhedra_Library::PIP_Solution_Node::row_sign (const Row & *x*, dimension_type *big_dimension*) [static, private]

Returns the sign of row `x`.

Definition at line 1461 of file PIP_Tree.cc.

References MIXED, NEGATIVE, Parma_Polyhedra_Library::not_a_dimension(), POSITIVE, sign, Parma_Polyhedra_Library::Row::size(), and ZERO.

Referenced by solve(), and update_tableau().

11.154.4.12 void Parma_Polyhedra_Library::PIP_Solution_Node::set_owner (const PIP_Problem * *owner*) [protected, virtual]

Sets the pointer to the PIP_Problem owning object.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 563 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::owner_.

Referenced by Parma_Polyhedra_Library::PIP_Problem::ascii_load().

11.154.4.13 PIP_Tree_Node * Parma_Polyhedra_Library::PIP_Solution_Node::solve (const PIP_Problem & *pip*, bool *check_feasible_context*, const Matrix & *context*, const Variables_Set & *params*, dimension_type *space_dim*, unsigned *indent_level*) [protected, virtual]

Implements pure virtual method PIP_Tree_Node::solve.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1887 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint(), Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Matrix::add_row(), Parma_Polyhedra_Library::PIP_Tree_Node::artificial_parameters, Parma_Polyhedra_Library::Matrix::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::begin(), big_dimension, Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::PIP_Tree_Node::constraints_, Parma_Polyhedra_Library::PIP_Problem::control_parameters, Parma_Polyhedra_Library::PIP_Problem::CUTTING_STRATEGY, Parma_Polyhedra_Library::PIP_Problem::CUTTING_STRATEGY_ALL, Parma_Polyhedra_Library::PIP_Problem::CUTTING_STRATEGY_DEEPEST, Parma_Polyhedra_Library::PIP_Problem::CUTTING_STRATEGY_FIRST, Parma_Polyhedra_Library::Constraint_System::empty(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::PIP_Decision_Node::false_child, Parma_Polyhedra_Library::gcd_assign(), generate_cut(), Parma_Polyhedra_Library::PIP_Tree_Node::get_owner(), Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::maybe_abandon(), MIXED, NEGATIVE, Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::PIP_Tree_Node::parent(), Parma_Polyhedra_Library::PIP_Tree_Node::PIP_Decision_Node, PIP_Solution_Node(), Parma_Polyhedra_Library::PIP_Problem::PIVOT_ROW_STRATEGY, Parma_Polyhedra_Library::PIP_Problem::PIVOT_ROW_STRATEGY_FIRST, POSITIVE, PPL_DIRTY_TEMP_COEFFICIENT, row_sign(), sign, solution_valid, Parma_Polyhedra_Library::PIP_Tree_Node::solve(), Parma_Polyhedra_Library::Constraint_System::swap(), Parma_Polyhedra_Library::Row::swap(), UNKNOWN, var_column, var_row, WEIGHT_ADD, WEIGHT_ADD_MUL, WEIGHT_BEGIN, and ZERO.

11.154.4.14 memory_size_type Parma_Polyhedra_Library::PIP_Solution_Node::total_memory_in_bytes () const [virtual]

Returns the total size in bytes of the memory occupied by `*this`.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 2903 of file PIP_Tree.cc.

References external_memory_in_bytes().

11.154.4.15 void Parma_Polyhedra_Library::PIP_Solution_Node::update_solution () const [protected]

Helper method.

Definition at line 3051 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::PIP_Tree_Node::get_owner(), Parma_Polyhedra_Library::PIP_-Problem::parameter_space_dimensions(), solution_valid, and Parma_Polyhedra_Library::PIP_-Problem::space_dimension().

Referenced by parametric_values(), and print_tree().

11.154.4.16 void Parma_Polyhedra_Library::PIP_Solution_Node::update_solution (const std::vector< bool > & *pip_dim_is_param*) const [protected]

Update the solution values.

Parameters

pip_dim_is_param A vector of Boolean flags telling which PIP problem dimensions are problem parameters. The size of the vector is equal to the PIP problem internal space dimension (i.e., no artificial parameters).

Definition at line 3069 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::add_mul_assign(), PPL_DIRTY_TEMP_COEFFICIENT, solution, and solution_valid.

11.154.4.17 void Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau (const PIP_Problem & *pip*, dimension_type *external_space_dim*, dimension_type *first_pending_constraint*, const Constraint_Sequence & *input_cs*, const Variables_Set & *parameters*) [protected, virtual]

Implements pure virtual method PIP_Tree_Node::update_tableau.

Implements Parma_Polyhedra_Library::PIP_Tree_Node.

Definition at line 1703 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::add_mul_assign(), big_dimension, Parma_Polyhedra_-Library::PIP_Problem::big_parameter_dimension, Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::PIP_Problem::external_space_dim, Parma_Polyhedra_-Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::PIP_Problem::internal_space_-dim, Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_-strict_inequality(), Parma_Polyhedra_Library::not_a_dimension(), OK(), row_sign(), sign, Parma_-

Polyhedra_Library::Constraint::space_dimension(), special_equality_row, var_column, var_row, WEIGHT_ADD, WEIGHT_BEGIN, and ZERO.

11.154.5 Friends And Related Function Documentation

11.154.5.1 bool PIP_Problem::ascii_load (std::istream & *s*) [friend]

11.154.6 Member Data Documentation

11.154.6.1 std::vector<bool> Parma_Polyhedra_Library::PIP_Solution_Node::basis [private]

A boolean vector for identifying the basic variables.

Variable identifiers are numbered from 0 to `n+m-1`, where `n` is the number of columns in the simplex tableau corresponding to variables, and `m` is the number of rows.

Indices from 0 to `n-1` correspond to the original variables.

Indices from `n` to `n+m-1` correspond to the slack variables associated to the internal constraints, which do not strictly correspond to original constraints, since these may have been transformed to fit the standard form of the dual simplex.

The value for `basis[i]` is:

- **true** if variable `i` is basic,
- **false** if variable `i` is nonbasic.

Definition at line 523 of file PIP_Tree.defs.hh.

11.154.6.2 dimension_type Parma_Polyhedra_Library::PIP_Solution_Node::big_dimension [private]

The column index in the parametric part of the simplex tableau corresponding to the big parameter; `not_a_dimension()` if not set.

Definition at line 570 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), solve(), and update_tableau().

11.154.6.3 std::vector<dimension_type> Parma_Polyhedra_Library::PIP_Solution_Node::mapping [private]

A mapping between the tableau rows/columns and the original variables.

The value of `mapping[i]` depends of the value of `basis[i]`.

- If `basis[i]` is **true**, `mapping[i]` encodes the column index of variable `i` in the `s` matrix of the tableau.
- If `basis[i]` is **false**, `mapping[i]` encodes the row index of variable `i` in the tableau.

Definition at line 535 of file PIP_Tree.defs.hh.

11.154.6.4 std::vector<Row_Sign> Parma_Polyhedra_Library::PIP_Solution_Node::sign [private]

A cache for computed sign values of constraint parametric RHS.

Definition at line 587 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), generate_cut(), row_sign(), solve(), and update_tableau().

11.154.6.5 std::vector<Linear_Expression> Parma_Polyhedra_Library::PIP_Solution_Node::solution [private]

Parametric values for the solution.

Definition at line 590 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), parametric_values(), print_tree(), and update_solution().

11.154.6.6 bool Parma_Polyhedra_Library::PIP_Solution_Node::solution_valid [private]

An indicator for solution validity.

Definition at line 593 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), solve(), and update_solution().

11.154.6.7 dimension_type Parma_Polyhedra_Library::PIP_Solution_Node::special_equality_row [private]

The variable number of the special inequality used for modelling equality constraints.

The subset of equality constraints in a specific problem can be expressed as: $f_i(x,p) = 0; 1 \leq i \leq n$. As the dual simplex standard form requires constraints to be inequalities, the following constraints can be modelized the following way:

- $f_i(x,p) \geq 0; 1 \leq i \leq n$
- $\sum_{i=1}^{n} f_i(x,p) \leq 0$

The `special_equality_row` value stores the variable number of the specific constraint which is used to modelize the latter sum of constraints. If no such constraint exists, the value is set to `0`.

Definition at line 564 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), and update_tableau().

11.154.6.8 Tableau Parma_Polyhedra_Library::PIP_Solution_Node::tableau [private]

The parametric simplex tableau.

Definition at line 503 of file PIP_Tree.defs.hh.

11.154.6.9 std::vector<dimension_type> Parma_Polyhedra_Library::PIP_Solution_Node::var_column [private]

The variable identifiers associated to the columns of the simplex tableau.

Definition at line 545 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), OK(), solve(), and update_tableau().

11.154.6.10 std::vector<dimension_type> Parma_Polyhedra_Library::PIP_Solution_Node::var_row [private]

The variable identifiers associated to the rows of the simplex tableau.

Definition at line 540 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), generate_cut(), OK(), solve(), and update_tableau().

The documentation for this class was generated from the following files:

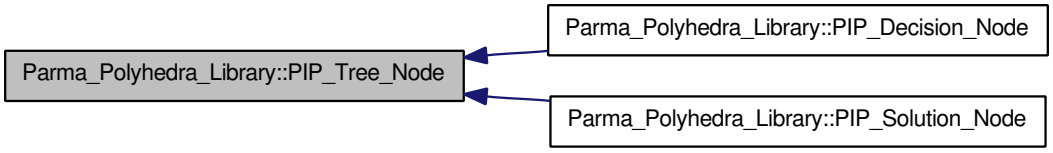
- PIP_Tree.defs.hh
- PIP_Tree.cc

11.155 Parma_Polyhedra_Library::PIP_Tree_Node Class Reference

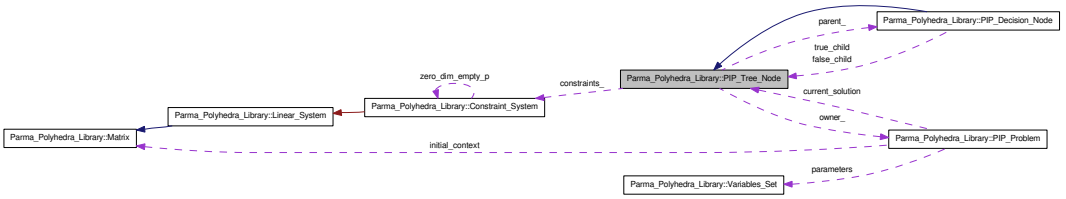
A node of the PIP solution tree.

`#include <PIP_Tree.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::PIP_Tree_Node:



Collaboration diagram for Parma_Polyhedra_Library::PIP_Tree_Node:



Classes

- class Artificial_Parameter

  Artificial parameters in PIP solution trees.

Public Types

- typedef std::vector< Artificial_Parameter > Artificial_Parameter_Sequence

  A type alias for a sequence of Artificial_Parameter's.

Public Member Functions

- virtual PIP_Tree_Node * clone () const =0

  Returns a pointer to a dynamically-allocated copy of `*this`.

- virtual ~PIP_Tree_Node ()

  Destructor.

- virtual bool OK () const

  Returns `true` *if and only if* `*this` *is well formed.*

- virtual const PIP_Solution_Node * as_solution () const

  Returns `this` *if* `*this` *is a solution node, 0 otherwise.*

- virtual const PIP_Decision_Node * as_decision () const

  Returns `this` *if* `*this` *is a decision node, 0 otherwise.*

- const Constraint_System & constraints () const

  Returns the system of parameter constraints controlling `*this`.

- Artificial_Parameter_Sequence::const_iterator art_parameter_begin () const

  Returns a const_iterator to the beginning of local artificial parameters.

- Artificial_Parameter_Sequence::const_iterator art_parameter_end () const

  Returns a const_iterator to the end of local artificial parameters.

- dimension_type art_parameter_count () const

Returns the number of local artificial parameters.

- void print (std::ostream &s, unsigned indent=0) const

  Prints on `s` *the tree rooted in* `*this`.

- void ascii_dump (std::ostream &s) const

  Dumps to `s` *an ASCII representation of* `*this`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- virtual memory_size_type total_memory_in_bytes () const =0

  Returns the total size in bytes of the memory occupied by `*this`.

- virtual memory_size_type external_memory_in_bytes () const =0

  Returns the size in bytes of the memory managed by `*this`.

Protected Types

- typedef std::vector< Constraint > Constraint_Sequence

  A type alias for a sequence of constraints.

Protected Member Functions

- PIP_Tree_Node (const PIP_Problem *owner)

  Constructor: builds a node owned by `*owner`.

- PIP_Tree_Node (const PIP_Tree_Node &y)

  Copy constructor.

- const PIP_Problem * get_owner () const

  Returns a pointer to the PIP_Problem owning object.

- virtual void set_owner (const PIP_Problem *owner)=0

  Sets the pointer to the PIP_Problem owning object.

- virtual bool check_ownership (const PIP_Problem *owner) const =0

  Returns `true` *if and only if all the nodes in the subtree rooted in* `*this` *is owned by* `*pip`.

- const PIP_Decision_Node * parent () const

  Returns a pointer to this node's parent.

- void set_parent (const PIP_Decision_Node *p)

  Set this node's parent to `*p`.

- virtual void update_tableau (const PIP_Problem &pip, dimension_type external_space_dim, dimension_type first_pending_constraint, const Constraint_Sequence &input_cs, const Variables_-Set &parameters)=0

  Populates the parametric simplex tableau using external data.

- virtual PIP_Tree_Node * solve (const PIP_Problem &pip, bool check_feasible_context, const Matrix &context, const Variables_Set &params, dimension_type space_dim, unsigned indent_level)=0

  Executes a parametric simplex on the tableau, under specified context.

- void add_constraint (const Row &x, const Variables_Set &parameters)

  Inserts a new parametric constraint in internal Row format.

- void parent_merge ()

  Merges parent's artificial parameters into `*this`.

- virtual void print_tree (std::ostream &s, unsigned indent, const std::vector< bool > &pip_dim_is_-param, dimension_type first_art_dim) const

  Prints on `s` *the tree rooted in* `*this`.

Static Protected Member Functions

- static void indent_and_print (std::ostream &s, unsigned indent, const char *str)

  A helper function used when printing PIP trees.

- static bool compatibility_check (Matrix &s)

  Checks whether a context matrix is satisfiable.

- static bool compatibility_check (const Matrix &context, const Row &row)

  Helper method: checks for satisfiability of the restricted context obtained by adding `row` *to* `context`.

Protected Attributes

- const PIP_Problem * owner_

  A pointer to the PIP_Problem object owning this node.

- const PIP_Decision_Node * parent_

  A pointer to the parent of `*this`*, null if* `*this` *is the root.*

- Constraint_System constraints_

  The local system of parameter constraints.

- Artificial_Parameter_Sequence artificial_parameters

  The local sequence of expressions for local artificial parameters.

Friends

- class PIP_Problem
- class PIP_Decision_Node
- class PIP_Solution_Node

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &os, const PIP_Tree_Node &x)

  Output operator: prints the solution tree rooted in `x`.

11.155.1 Detailed Description

A node of the PIP solution tree. This is the base class for the nodes of the binary trees representing the solutions of PIP problems. From this one, two classes are derived:

- PIP_Decision_Node, for the internal nodes of the tree;
- PIP_Solution_Node, for the leaves of the tree.

Definition at line 47 of file PIP_Tree.defs.hh.

11.155.2 Member Typedef Documentation

11.155.2.1 typedef std::vector<Artificial_Parameter> Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter_Sequence

A type alias for a sequence of Artificial_Parameter's.

Definition at line 92 of file PIP_Tree.defs.hh.

11.155.2.2 typedef std::vector<Constraint> Parma_Polyhedra_Library::PIP_Tree_Node::Constraint_Sequence [protected]

A type alias for a sequence of constraints.

Definition at line 133 of file PIP_Tree.defs.hh.

11.155.3 Constructor & Destructor Documentation

11.155.3.1 Parma_Polyhedra_Library::PIP_Tree_Node::PIP_Tree_Node (const PIP_Problem * *owner*) [explicit, protected]

Constructor: builds a node owned by `*owner`.

Definition at line 355 of file PIP_Tree.cc.

11.155.3.2 Parma_Polyhedra_Library::PIP_Tree_Node::PIP_Tree_Node (const PIP_Tree_Node & *y*) [protected]

Copy constructor.

Definition at line 362 of file PIP_Tree.cc.

11.155.3.3 Parma_Polyhedra_Library::PIP_Tree_Node::~PIP_Tree_Node () [inline, virtual]

Destructor.

Definition at line 56 of file PIP_Tree.inlines.hh.

11.155.4 Member Function Documentation

11.155.4.1 void Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint (const Row & *x*, const Variables_Set & *parameters*) [protected]

Inserts a new parametric constraint in internal Row format.

Definition at line 663 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Row::size(), WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::solve().

11.155.4.2 PIP_Tree_Node::Artificial_Parameter_Sequence::const_iterator Parma_Polyhedra_Library::PIP_Tree_Node::art_parameter_begin () const [inline]

Returns a const_iterator to the beginning of local artificial parameters.

Definition at line 80 of file PIP_Tree.inlines.hh.

References artificial_parameters.

Referenced by external_memory_in_bytes(), parent_merge(), and print_tree().

11.155.4.3 dimension_type Parma_Polyhedra_Library::PIP_Tree_Node::art_parameter_count () const [inline]

Returns the number of local artificial parameters.

Definition at line 90 of file PIP_Tree.inlines.hh.

References artificial_parameters.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::print_tree().

11.155.4.4 PIP_Tree_Node::Artificial_Parameter_Sequence::const_iterator Parma_Polyhedra_Library::PIP_Tree_Node::art_parameter_end () const [inline]

Returns a const_iterator to the end of local artificial parameters.

Definition at line 85 of file PIP_Tree.inlines.hh.

References artificial_parameters.

Referenced by external_memory_in_bytes(), parent_merge(), and print_tree().

11.155.4.5 const PIP_Decision_Node * Parma_Polyhedra_Library::PIP_Tree_Node::as_decision () const [virtual]

Returns `this` if `*this` is a decision node, 0 otherwise.

Reimplemented in Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 594 of file PIP_Tree.cc.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::ascii_dump(), and Parma_Polyhedra_-Library::PIP_Problem::ascii_dump().

11.155.4.6 const PIP_Solution_Node * Parma_Polyhedra_Library::PIP_Tree_Node::as_solution () const [virtual]

Returns `this` if `*this` is a solution node, 0 otherwise.

Reimplemented in Parma_Polyhedra_Library::PIP_Solution_Node.

Definition at line 589 of file PIP_Tree.cc.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::ascii_dump(), and Parma_Polyhedra_-Library::PIP_Problem::ascii_dump().

11.155.4.7 void Parma_Polyhedra_Library::PIP_Tree_Node::ascii_dump (std::ostream & *s*) const

Dumps to `s` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_-Decision_Node.

Definition at line 1186 of file PIP_Tree.cc.

References artificial_parameters, Parma_Polyhedra_Library::Constraint_System::ascii_dump(), and constraints_.

Referenced by OK().

11.155.4.8 bool Parma_Polyhedra_Library::PIP_Tree_Node::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 1196 of file PIP_Tree.cc.

References artificial_parameters, Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::ascii_load(), Parma_Polyhedra_Library::Constraint_System::ascii_load(), and constraints_.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::ascii_load(), and Parma_Polyhedra_Library::PIP_Decision_Node::ascii_load().

11.155.4.9 virtual bool Parma_Polyhedra_Library::PIP_Tree_Node::check_ownership (const PIP_Problem * *owner*) const [protected, pure virtual]

Returns `true` if and only if all the nodes in the subtree rooted in `*this` is owned by `*pip`.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::check_ownership(), and Parma_Polyhedra_Library::PIP_Problem::OK().

11.155.4.10 virtual PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Tree_Node::clone () const [pure virtual]

Returns a pointer to a dynamically-allocated copy of `*this`.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::PIP_Decision_Node(), and Parma_Polyhedra_Library::PIP_Problem::PIP_Problem().

11.155.4.11 bool Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check (const Matrix & *context*, const Row & *row*) [static, protected]

Helper method: checks for satisfiability of the restricted context obtained by adding `row` to `context`.

Definition at line 1492 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Matrix::add_row(), and compatibility_check().

11.155.4.12 bool Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check (Matrix & *s*) [static, protected]

Checks whether a context matrix is satisfiable.

The satisfiability check is implemented by the revised dual simplex algorithm on the context matrix. The algorithm ensures the feasible solution is integer by applying a cut generation method when intermediate non-integer solutions are found.

Definition at line 1500 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::maybe_abandon(), Parma_Polyhedra_Library::not_a_dimension(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::swap(), WEIGHT_ADD, WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by compatibility_check(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_Decision_Node::solve(), and Parma_Polyhedra_Library::PIP_Problem::solve().

11.155.4.13 const Constraint_System & Parma_Polyhedra_Library::PIP_Tree_Node::constraints () const [inline]

Returns the system of parameter constraints controlling `*this`.

The indices in the constraints are the same as the original variables and parameters. Coefficients in indices corresponding to variables always are zero.

Definition at line 75 of file PIP_Tree.inlines.hh.

References constraints_.

11.155.4.14 memory_size_type Parma_Polyhedra_Library::PIP_Tree_Node::external_memory_in_bytes () const [pure virtual]

Returns the size in bytes of the memory managed by `*this`.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 2850 of file PIP_Tree.cc.

References art_parameter_begin(), art_parameter_end(), artificial_parameters, constraints_, and Parma_Polyhedra_Library::Constraint_System::external_memory_in_bytes().

11.155.4.15 const PIP_Problem * Parma_Polyhedra_Library::PIP_Tree_Node::get_owner () const [inline, protected]

Returns a pointer to the PIP_Problem owning object.

Definition at line 70 of file PIP_Tree.inlines.hh.

References owner_.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::check_ownership(), Parma_Polyhedra_Library::PIP_Solution_Node::check_ownership(), Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values(), print(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_solution().

11.155.4.16 void Parma_Polyhedra_Library::PIP_Tree_Node::indent_and_print (std::ostream & *s*, unsigned *indent*, const char * *str*) [static, protected]

A helper function used when printing PIP trees.

Definition at line 2908 of file PIP_Tree.cc.

Referenced by Parma_Polyhedra_Library::PIP_Problem::print_solution(), Parma_Polyhedra_Library::PIP_Solution_Node::print_tree(), Parma_Polyhedra_Library::PIP_Decision_Node::print_tree(), print_tree(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.4.17 bool Parma_Polyhedra_Library::PIP_Tree_Node::OK () const [virtual]

Returns `true` if and only if `*this` is well formed.

Reimplemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 637 of file PIP_Tree.cc.

References ascii_dump(), Parma_Polyhedra_Library::Constraint_System::begin(), constraints_, and Parma_Polyhedra_Library::Constraint_System::end().

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::OK(), Parma_Polyhedra_Library::PIP_Solution_Node::OK(), Parma_Polyhedra_Library::PIP_Problem::OK(), parent_merge(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.4.18 const PIP_Decision_Node * Parma_Polyhedra_Library::PIP_Tree_Node::parent () const [inline, protected]

Returns a pointer to this node's parent.

Definition at line 65 of file PIP_Tree.inlines.hh.

References parent_.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), parent_merge(), print(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.4.19 void Parma_Polyhedra_Library::PIP_Tree_Node::parent_merge () [protected]

Merges parent's artificial parameters into `*this`.

Definition at line 685 of file PIP_Tree.cc.

References art_parameter_begin(), art_parameter_end(), artificial_parameters, OK(), parent(), and parent_.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.4.20 void Parma_Polyhedra_Library::PIP_Tree_Node::print (std::ostream & *s*, unsigned *indent* = 0) const

Prints on `s` the tree rooted in `*this`.

Parameters

s The output stream.

indent The amount of indentation.

Definition at line 2915 of file PIP_Tree.cc.

References get_owner(), Parma_Polyhedra_Library::PIP_Problem::parameter_space_dimensions(), parent(), print_tree(), and Parma_Polyhedra_Library::PIP_Problem::space_dimension().

Referenced by Parma_Polyhedra_Library::IO_Operators::operator<<(), and Parma_Polyhedra_Library::PIP_Problem::print_solution().

11.155.4.21 void Parma_Polyhedra_Library::PIP_Tree_Node::print_tree (std::ostream & *s*, unsigned *indent*, const std::vector< bool > & *pip_dim_is_param*, dimension_type *first_art_dim*) const [protected, virtual]

Prints on `s` the tree rooted in `*this`.

Parameters

s The output stream.

indent The amount of indentation.

pip_dim_is_param A vector of Boolean flags telling which PIP problem dimensions are problem parameters. The size of the vector is equal to the PIP problem internal space dimension (i.e., no artificial parameters).

first_art_dim The first space dimension corresponding to an artificial parameter that was created in this node (if any).

Reimplemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 2932 of file PIP_Tree.cc.

References art_parameter_begin(), art_parameter_end(), Parma_Polyhedra_Library::Constraint_System::begin(), constraints_, Parma_Polyhedra_Library::Constraint_System::empty(), Parma_Polyhedra_Library::Constraint_System::end(), and indent_and_print().

Referenced by print(), and Parma_Polyhedra_Library::PIP_Decision_Node::print_tree().

11.155.4.22 virtual void Parma_Polyhedra_Library::PIP_Tree_Node::set_owner (const PIP_Problem * *owner*) [protected, pure virtual]

Sets the pointer to the PIP_Problem owning object.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_-Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Problem::PIP_Problem(), and Parma_Polyhedra_-Library::PIP_Decision_Node::set_owner().

11.155.4.23 void Parma_Polyhedra_Library::PIP_Tree_Node::set_parent (const PIP_Decision_Node * *p*) [inline, protected]

Set this node's parent to *p.

Definition at line 60 of file PIP_Tree.inlines.hh.

References parent_.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::PIP_Decision_Node(), and Parma_-Polyhedra_Library::PIP_Decision_Node::solve().

11.155.4.24 virtual PIP_Tree_Node* Parma_Polyhedra_Library::PIP_Tree_Node::solve (const PIP_Problem & *pip*, bool *check_feasible_context*, const Matrix & *context*, const Variables_Set & *params*, dimension_type *space_dim*, unsigned *indent_level*) [protected, pure virtual]

Executes a parametric simplex on the tableau, under specified context.

Returns

The root of the PIP tree solution, or 0 if unfeasible.

Parameters

pip The PIP_Problem object containing this node.

check_feasible_context Whether the resolution process should (re-)check feasibility of context (since the initial context may have been modified).

context The context, being a set of constraints on the parameters.

params The local parameter set, including parent's artificial parameters.

space_dim The space dimension of parent, including artificial parameters.

indent_level The indentation level (for debugging output only).

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_-Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::PIP_-Decision_Node::solve(), and Parma_Polyhedra_Library::PIP_Problem::solve().

11.155.4.25 virtual memory_size_type Parma_Polyhedra_Library::PIP_Tree_Node::total_memory_in_bytes () const [pure virtual]

Returns the total size in bytes of the memory occupied by `*this`.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Decision_Node::external_memory_in_bytes(), and Parma_Polyhedra_Library::PIP_Problem::external_memory_in_bytes().

11.155.4.26 virtual void Parma_Polyhedra_Library::PIP_Tree_Node::update_tableau (const PIP_Problem & *pip*, dimension_type *external_space_dim*, dimension_type *first_pending_constraint*, const Constraint_Sequence & *input_cs*, const Variables_Set & *parameters*) [protected, pure virtual]

Populates the parametric simplex tableau using external data.

Parameters

pip The PIP_Problem object containing this node.

external_space_dim The number of all problem variables and problem parameters (excluding artificial parameters).

first_pending_constraint The first element in `input_cs` to be added to the tableau, which already contains the previous elements.

input_cs All the constraints of the PIP problem.

parameters The set of indices of the problem parameters.

Implemented in Parma_Polyhedra_Library::PIP_Solution_Node, and Parma_Polyhedra_Library::PIP_Decision_Node.

Referenced by Parma_Polyhedra_Library::PIP_Problem::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::update_tableau().

11.155.5 Friends And Related Function Documentation

11.155.5.1 std::ostream & operator<< (std::ostream & *os*, const PIP_Tree_Node & *x*) [related]

Output operator: prints the solution tree rooted in `x`.

Definition at line 341 of file PIP_Tree.cc.

11.155.5.2 friend class PIP_Decision_Node [friend]

Definition at line 138 of file PIP_Tree.defs.hh.

Referenced by Parma_Polyhedra_Library::PIP_Solution_Node::solve().

11.155.5.3 friend class PIP_Problem [friend]

Definition at line 137 of file PIP_Tree.defs.hh.

11.155.5.4 friend class PIP_Solution_Node [friend]

Reimplemented in Parma_Polyhedra_Library::PIP_Decision_Node.

Definition at line 139 of file PIP_Tree.defs.hh.

11.155.6 Member Data Documentation

11.155.6.1 Artificial_Parameter_Sequence Parma_Polyhedra_Library::PIP_Tree_Node::artificial_parameters [protected]

The local sequence of expressions for local artificial parameters.

Definition at line 151 of file PIP_Tree.defs.hh.

Referenced by art_parameter_begin(), art_parameter_count(), art_parameter_end(), ascii_dump(), ascii_load(), external_memory_in_bytes(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), parent_merge(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.6.2 Constraint_System Parma_Polyhedra_Library::PIP_Tree_Node::constraints_ [protected]

The local system of parameter constraints.

Definition at line 148 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), constraints(), external_memory_in_bytes(), Parma_Polyhedra_Library::PIP_Decision_Node::OK(), OK(), Parma_Polyhedra_Library::PIP_Solution_Node::print_tree(), print_tree(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Decision_Node::solve().

11.155.6.3 const PIP_Problem* Parma_Polyhedra_Library::PIP_Tree_Node::owner_ [protected]

A pointer to the PIP_Problem object owning this node.

Definition at line 142 of file PIP_Tree.defs.hh.

Referenced by get_owner(), Parma_Polyhedra_Library::PIP_Decision_Node::set_owner(), and Parma_Polyhedra_Library::PIP_Solution_Node::set_owner().

11.155.6.4 const PIP_Decision_Node* Parma_Polyhedra_Library::PIP_Tree_Node::parent_ [protected]

A pointer to the parent of `*this`, null if `*this` is the root.

Definition at line 145 of file PIP_Tree.defs.hh.

Referenced by parent(), parent_merge(), and set_parent().

The documentation for this class was generated from the following files:

- PIP_Tree.defs.hh
- PIP_Tree.inlines.hh
- PIP_Tree.cc

11.156 Parma_Polyhedra_Library::Plus_Infinity Struct Reference

`#include <checked.defs.hh>`

Static Public Attributes

- static const Result_Class vclass = VC_PLUS_INFINITY

11.156.1 Detailed Description

Definition at line 543 of file checked.defs.hh.

11.156.2 Member Data Documentation

11.156.2.1 const Result_Class Parma_Polyhedra_Library::Plus_Infinity::vclass = VC_PLUS_INFINITY [static]

Definition at line 544 of file checked.defs.hh.

The documentation for this struct was generated from the following file:

- checked.defs.hh

11.157 Parma_Polyhedra_Library::Pointset_Powerset< PSET > Class Template Reference

The powerset construction instantiated on PPL pointset domains.

`#include <Pointset_Powerset.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Pointset_Powerset< PSET >:

Collaboration diagram for Parma_Polyhedra_Library::Pointset_Powerset< PSET >:

Public Types

- typedef PSET element_type
- typedef Base::size_type size_type
- typedef Base::value_type value_type
- typedef Base::iterator iterator

  Alias for a read-only bidirectional iterator on the disjuncts of a Powerset element.

- typedef Base::const_iterator const_iterator

  A bidirectional const_iterator on the disjuncts of a Powerset element.

- typedef Base::reverse_iterator reverse_iterator

  The reverse iterator type built from Powerset::iterator.

- typedef Base::const_reverse_iterator const_reverse_iterator

  The reverse iterator type built from Powerset::const_iterator.

Public Member Functions

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- template<>
  Pointset_Powerset (const Pointset_Powerset< QH > &y, Complexity_Class)
- template<>
  Pointset_Powerset (const Pointset_Powerset< C_Polyhedron > &y, Complexity_Class)
- template<>
  Pointset_Powerset (const Pointset_Powerset< Grid > &y, Complexity_Class)
- template<>
  Pointset_Powerset (const Pointset_Powerset< NNC_Polyhedron > &y, Complexity_Class)
- template<>
  void difference_assign (const Pointset_Powerset &y)
- template<>
  void difference_assign (const Pointset_Powerset &y)

- template<>
  bool geometrically_covers (const Pointset_Powerset &y) const
- template<>
  bool geometrically_covers (const Pointset_Powerset &y) const
- template<>
  bool geometrically_equals (const Pointset_Powerset &y) const
- template<>
  bool geometrically_equals (const Pointset_Powerset &y) const
- template<>
  Pointset_Powerset (const Pointset_Powerset< QH > &y, Complexity_Class complexity)
- template<>
  void difference_assign (const Pointset_Powerset &y)
- template<>
  bool geometrically_covers (const Pointset_Powerset &y) const
- template<>
  void difference_assign (const Pointset_Powerset &y)
- template<>
  bool geometrically_covers (const Pointset_Powerset &y) const
- template<>
  Pointset_Powerset (const Pointset_Powerset< C_Polyhedron > &y, Complexity_Class)
- template<>
  Pointset_Powerset (const Pointset_Powerset< Grid > &y, Complexity_Class)
- template<>
  Pointset_Powerset (const Pointset_Powerset< NNC_Polyhedron > &y, Complexity_Class)

Constructors

- Pointset_Powerset (dimension_type num_dimensions=0, Degenerate_Element kind=UNIVERSE)

  Builds a universe (top) or empty (bottom) Pointset_Powerset.

- Pointset_Powerset (const Pointset_Powerset &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- template<typename QH >
  Pointset_Powerset (const Pointset_Powerset< QH > &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Conversion constructor: the type `QH` of the disjuncts in the source powerset is different from `PSET`.

- template<typename QH1 , typename QH2 , typename R >
  Pointset_Powerset (const Partially_Reduced_Product< QH1, QH2, R > &prp, Complexity_Class complexity=ANY_COMPLEXITY)

  Creates a Pointset_Powerset from a product This will be created as a single disjunct of type PSET that approximates the product.

- Pointset_Powerset (const Constraint_System &cs)

  Creates a Pointset_Powerset with a single disjunct approximating the system of constraints `cs`.

- Pointset_Powerset (const Congruence_System &cgs)

  Creates a Pointset_Powerset with a single disjunct approximating the system of congruences `cgs`.

- Pointset_Powerset (const C_Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)

Builds a pointset_powerset out of a closed polyhedron.

- Pointset_Powerset (const NNC_Polyhedron &ph, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a pointset_powerset out of an nnc polyhedron.

- Pointset_Powerset (const Grid &gr, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a pointset_powerset out of a grid.

- template<typename T >
  Pointset_Powerset (const Octagonal_Shape< T > &os, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a pointset_powerset out of an octagonal shape.

- template<typename T >
  Pointset_Powerset (const BD_Shape< T > &bds, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a pointset_powerset out of a bd shape.

- template<typename Interval >
  Pointset_Powerset (const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)
  Builds a pointset_powerset out of a box.

Member Functions that Do Not Modify the Pointset_Powerset

- dimension_type space_dimension () const
  Returns the dimension of the vector space enclosing `*this`.

- dimension_type affine_dimension () const
  Returns the dimension of the vector space enclosing `*this`.

- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty powerset.*

- bool is_universe () const
  Returns `true` *if and only if* `*this` *is the top element of the powerser lattice.*

- bool is_topologically_closed () const
  Returns `true` *if and only if all the disjuncts in* `*this` *are topologically closed.*

- bool is_bounded () const
  Returns `true` *if and only if all elements in* `*this` *are bounded.*

- bool is_disjoint_from (const Pointset_Powerset &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*

- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*

- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.

- bool bounds_from_above (const Linear_Expression &expr) const

Returns `true` *if and only if* `expr` *is bounded from above in* `*this`.

- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from below in* `*this`.

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool geometrically_covers (const Pointset_Powerset &y) const
  Returns `true` *if and only if* `*this` *geometrically covers* `y`*, i.e., if any point (in some element) of* `y` *is also a point (of some element) of* `*this`.

- bool geometrically_equals (const Pointset_Powerset &y) const
  Returns `true` *if and only if* `*this` *is geometrically equal to* `y`*, i.e., if (the elements of)* `*this` *and* `y` *contain the same set of points.*

- bool contains (const Pointset_Powerset &y) const
  Returns `true` *if and only if each disjunct of* `y` *is contained in a disjunct of* `*this`.

- bool strictly_contains (const Pointset_Powerset &y) const
  Returns `true` *if and only if each disjunct of* `y` *is strictly contained in a disjunct of* `*this`.

- bool contains_integer_point () const
  Returns `true` *if and only if* `*this` *contains at least one integer point.*

- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between the powerset `*this` *and the constraint* `c`.

- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between the powerset `*this` *and the generator* `g`.

- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between the powerset `*this` *and the congruence* `c`.

- memory_size_type total_memory_in_bytes () const
  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const
  Returns a lower bound to the size in bytes of the memory managed by `*this`.

- int32_t hash_code () const
  Returns a 32-bit hash code for `*this`.

- bool OK () const
  Checks if all the invariants are satisfied.

Space Dimension Preserving Member Functions that May Modify the Pointset_Powerset

- void add_disjunct (const PSET &ph)
  Adds to `*this` *the disjunct* `ph`.

- void add_constraint (const Constraint &c)
  Intersects `*this` *with constraint* `c`.

- void refine_with_constraint (const Constraint &c)
  Use the constraint `c` *to refine* `*this`.

- void add_constraints (const Constraint_System &cs)
  Intersects `*this` *with the constraints in* `cs`.

- void refine_with_constraints (const Constraint_System &cs)
  Use the constraints in `cs` *to refine* `*this`.

- void add_congruence (const Congruence &c)
  Intersects `*this` *with congruence* `c`.

- void refine_with_congruence (const Congruence &cg)
  Use the congruence `cg` *to refine* `*this`.

- void add_congruences (const Congruence_System &cgs)
  Intersects `*this` *with the congruences in* `cgs`.

- void refine_with_congruences (const Congruence_System &cgs)
  Use the congruences in `cgs` *to refine* `*this`.

- void unconstrain (Variable var)
  Computes the cylindrification of `*this` *with respect to space dimension* `var`*, assigning the result to* `*this`.

- void unconstrain (const Variables_Set &vars)
  Computes the cylindrification of `*this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `*this`.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)
  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)
  Possibly tightens `*this` *by dropping some points with non-integer coordinates for the space dimensions corresponding to* `vars`.

- void topological_closure_assign ()
  Assigns to `*this` *its topological closure.*

- void intersection_assign (const Pointset_Powerset &y)

  Assigns to `*this` *the intersection of* `*this` *and* `y`.

- void difference_assign (const Pointset_Powerset &y)

  Assigns to `*this` *an (a smallest) over-approximation as a powerset of the disjunct domain of the set-theoretical difference of* `*this` *and* `y`.

- bool simplify_using_context_assign (const Pointset_Powerset &y)

  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`. *If* `false` *is returned, then the intersection is empty.*

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())

  Assigns to `*this` *the affine image of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_-reference denominator=Coefficient_one())

  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`.

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_-Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`.

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_-Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_-Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$.

- void time_elapse_assign (const Pointset_Powerset &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`.

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)

  Wraps the specified dimensions of the vector space.

- void pairwise_reduce ()

  Assign to `*this` the result of (recursively) merging together the pairs of disjuncts whose upper-bound is the same as their set-theoretical union.

- template<typename Widening >
  void BGP99_extrapolation_assign (const Pointset_Powerset &y, Widening wf, unsigned max_disjuncts)

  Assigns to `*this` the result of applying the BGP99 extrapolation operator to `*this` and `y`, using the widening function `wf` and the cardinality threshold `max_disjuncts`.

- template<typename Cert , typename Widening >
  void BHZ03_widening_assign (const Pointset_Powerset &y, Widening wf)

  Assigns to `*this` the result of computing the BHZ03-widening between `*this` and `y`, using the widening function `wf` certified by the convergence certificate `Cert`.

Member Functions that May Modify the Dimension of the Vector Space

- Pointset_Powerset & operator= (const Pointset_Powerset &y)

  The assignment operator (`*this` and `y` can be dimension-incompatible).

- template<typename QH >
  Pointset_Powerset & operator= (const Pointset_Powerset< QH > &y)

  Conversion assignment: the type `QH` of the disjuncts in the source powerset is different from `PSET` (`*this` and `y` can be dimension-incompatible).

- void swap (Pointset_Powerset &y)

  Swaps `*this` with `y`.

- void add_space_dimensions_and_embed (dimension_type m)

  Adds `m` new dimensions to the vector space containing `*this` and embeds each disjunct in `*this` in the new space.

- void add_space_dimensions_and_project (dimension_type m)

  Adds `m` new dimensions to the vector space containing `*this` without embedding the disjuncts in `*this` in the new space.

- void concatenate_assign (const Pointset_Powerset &y)

  Assigns to `*this` the concatenation of `*this` and `y`.

- void remove_space_dimensions (const Variables_Set &vars)

  Removes all the specified space dimensions.

- void remove_higher_space_dimensions (dimension_type new_dimension)

  Removes the higher space dimensions so that the resulting space will have dimension `new_dimension`.

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)

  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)

  Creates `m` *copies of the space dimension corresponding to* `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)

  Folds the space dimensions in `vars` *into* `dest`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Pointset_Powerset<PSET> can handle.

Private Types

- typedef Determinate< PSET > Det_PSET
- typedef Powerset< Det_PSET > Base
- typedef Base::Sequence Sequence

  A powerset is implemented as a sequence of elements.

- typedef Base::Sequence_iterator Sequence_iterator

  Alias for the low-level iterator on the disjuncts.

- typedef Base::Sequence_const_iterator Sequence_const_iterator

  Alias for the low-level const_iterator on the disjuncts.

Private Member Functions

- bool intersection_preserving_enlarge_element (PSET &dest) const

  Assigns to `dest` *a powerset meet-preserving enlargement of itself with respect to* `*this`. *If* `false` *is returned, then the intersection is empty.*

- template<typename Widening >
  void BGP99_heuristics_assign (const Pointset_Powerset &y, Widening wf)

  Assigns to `*this` *the result of applying the BGP99 heuristics to* `*this` *and* `y`, *using the widening function* `wf`.

- template<typename Cert >
  void collect_certificates (std::map< Cert, size_type, typename Cert::Compare > &cert_ms) const

  Records in `cert_ms` *the certificates for this set of disjuncts.*

- template<typename Cert >
  bool is_cert_multiset_stabilizing (const std::map< Cert, size_type, typename Cert::Compare > &y_cert_ms) const

  Returns `true` *if and only if the current set of dijsuncts is stabilizing with respect to the multiset of certificates* `y_cert_ms`.

Private Attributes

- dimension_type space_dim

  The number of dimensions of the enclosing vector space.

Friends

- class Pointset_Powerset< NNC_Polyhedron >

Related Functions

(Note that these are not member functions.)

- template<typename PSET >
  Widening_Function< PSET > widen_fun_ref (void(PSET::∗wm)(const PSET &, unsigned ∗))

  Wraps a widening method into a function object.

- template<typename PSET , typename CSYS >
  Limited_Widening_Function< PSET, CSYS > widen_fun_ref (void(PSET::∗lwm)(const PSET &, const CSYS &, unsigned ∗), const CSYS &cs)

  Wraps a limited widening method into a function object.

- template<typename PSET >
  std::pair< PSET, Pointset_Powerset< NNC_Polyhedron > > linear_partition (const PSET &p, const PSET &q)

  Partitions `q` *with respect to* `p`.

- bool check_containment (const NNC_Polyhedron &ph, const Pointset_Powerset< NNC_-Polyhedron > &ps)

  Returns `true` *if and only if the union of the NNC polyhedra in* `ps` *contains the NNC polyhedron* `ph`.

- std::pair< Grid, Pointset_Powerset< Grid > > approximate_partition (const Grid &p, const Grid &q, bool &finite_partition)

  Partitions the grid `q` *with respect to grid* `p` *if and only if such a partition is finite.*

- bool check_containment (const Grid &ph, const Pointset_Powerset< Grid > &ps)

  Returns `true` *if and only if the union of the grids* `ps` *contains the grid* `g`.

- template<typename PSET >
  bool check_containment (const PSET &ph, const Pointset_Powerset< PSET > &ps)

  Returns `true` *if and only if the union of the objects in* `ps` *contains* `ph`.

- template<typename PSET >
  void swap (Parma_Polyhedra_Library::Pointset_Powerset< PSET > &x, Parma_Polyhedra_-Library::Pointset_Powerset< PSET > &y)

  Specializes `std::swap`.

- template<>
  bool check_containment (const C_Polyhedron &ph, const Pointset_Powerset< C_Polyhedron > &ps)

- template<typename PSET >
  void linear_partition_aux (const Constraint &c, PSET &qq, Pointset_Powerset< NNC_Polyhedron > &r)

  Partitions polyhedron `qq` *according to constraint* `c`.

- bool approximate_partition_aux (const PPL::Congruence &c, PPL::Grid &qq, PPL::Pointset_Powerset< PPL::Grid > &r)

  Uses the congruence `c` *to approximately partition the grid* `qq`.

11.157.1 Detailed Description

template<typename PSET> class Parma_Polyhedra_Library::Pointset_Powerset< PSET >

The powerset construction instantiated on PPL pointset domains.

Warning

At present, the supported instantiations for the disjunct domain template `PSET` are the simple pointset domains: `C_Polyhedron`, `NNC_Polyhedron`, `Grid`, `Octagonal_Shape<T>`, `BD_Shape<T>`, `Box<T>`.

Definition at line 62 of file Pointset_Powerset.defs.hh.

11.157.2 Member Typedef Documentation

11.157.2.1 template<typename PSET> typedef Powerset<Det_PSET> Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Base [private]

Definition at line 70 of file Pointset_Powerset.defs.hh.

11.157.2.2 template<typename PSET> typedef Base::const_iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::const_iterator

A bidirectional const_iterator on the disjuncts of a Powerset element.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1245 of file Pointset_Powerset.defs.hh.

11.157.2.3 template<typename PSET> typedef Base::const_reverse_iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::const_reverse_iterator

The reverse iterator type built from Powerset::const_iterator.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1247 of file Pointset_Powerset.defs.hh.

11.157.2.4 template<typename PSET> typedef Determinate<PSET> Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Det_PSET [private]

Definition at line 69 of file Pointset_Powerset.defs.hh.

11.157.2.5 template<typename PSET> typedef PSET Parma_Polyhedra_Library::Pointset_Powerset< PSET >::element_type

Definition at line 66 of file Pointset_Powerset.defs.hh.

11.157.2.6 template<typename PSET> typedef Base::iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::iterator

Alias for a *read-only* bidirectional iterator on the disjuncts of a Powerset element.

By using this iterator type, the disjuncts cannot be overwritten, but they can be removed using methods `drop_disjunct(iterator position)` and `drop_disjuncts(iterator first, iterator last)`, while still ensuring a correct handling of Omega-reduction.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1244 of file Pointset_Powerset.defs.hh.

11.157.2.7 template<typename PSET> typedef Base::reverse_iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::reverse_iterator

The reverse iterator type built from Powerset::iterator.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1246 of file Pointset_Powerset.defs.hh.

11.157.2.8 template<typename PSET> typedef Base::Sequence Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Sequence [private]

A powerset is implemented as a sequence of elements.

The particular sequence employed must support efficient deletion in any position and efficient back insertion.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1259 of file Pointset_Powerset.defs.hh.

11.157.2.9 template<typename PSET> typedef Base::Sequence_const_iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Sequence_const_iterator [private]

Alias for the low-level const_iterator on the disjuncts.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1261 of file Pointset_Powerset.defs.hh.

11.157.2.10 template<typename PSET> typedef Base::Sequence_iterator Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Sequence_iterator [private]

Alias for the low-level iterator on the disjuncts.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1260 of file Pointset_Powerset.defs.hh.

11.157.2.11 template<typename PSET> typedef Base::size_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::size_type

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1242 of file Pointset_Powerset.defs.hh.

11.157.2.12 template<typename PSET> typedef Base::value_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::value_type

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 1243 of file Pointset_Powerset.defs.hh.

11.157.3 Constructor & Destructor Documentation

11.157.3.1 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (dimension_type *num_dimensions* = 0, Degenerate_Element *kind* = UNIVERSE) [inline, explicit]

Builds a universe (top) or empty (bottom) Pointset_Powerset.

Parameters

num_dimensions The number of dimensions of the vector space enclosing the powerset;

kind Specifies whether the universe or the empty powerset has to be built.

Definition at line 54 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::UNIVERSE.

11.157.3.2 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Pointset_Powerset< PSET > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 65 of file Pointset_Powerset.inlines.hh.

11.157.3.3 template<typename PSET > template<typename QH > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Pointset_Powerset< QH > & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Conversion constructor: the type `QH` of the disjuncts in the source powerset is different from `PSET`.

Parameters

y The powerset to be used to build the new powerset.

complexity The maximal complexity of any algorithms used.

Definition at line 87 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.4 template<typename PSET > template<typename QH1 , typename QH2 , typename R > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Partially_Reduced_Product< QH1, QH2, R > & *prp*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Creates a Pointset_Powerset from a product This will be created as a single disjunct of type PSET that approximates the product.

Definition at line 119 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_Library::Partially_-Reduced_Product< D1, D2, R >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.5 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Constraint_System & *cs*) [inline, explicit]

Creates a Pointset_Powerset with a single disjunct approximating the system of constraints `cs`.

Definition at line 169 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK().

11.157.3.6 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Congruence_System & *cgs*) [inline, explicit]

Creates a Pointset_Powerset with a single disjunct approximating the system of congruences `cgs`.

Definition at line 176 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK().

11.157.3.7 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const C_Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of a closed polyhedron.

Builds a powerset that is either empty (if the polyhedron is found to be empty) or contains a single disjunct approximating the polyhedron; this must only use algorithms that do not exceed the specified complexity. The powerset inherits the space dimension of the polyhedron.

Parameters

ph The closed polyhedron to be used to build the powerset.

complexity The maximal complexity of any algorithms used.

Exceptions

std::length_error Thrown if the space dimension of `ph` exceeds the maximum allowed space dimension.

Definition at line 72 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_-Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.8 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const NNC_Polyhedron & *ph*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of an nnc polyhedron.

Builds a powerset that is either empty (if the polyhedron is found to be empty) or contains a single disjunct approximating the polyhedron; this must only use algorithms that do not exceed the specified complexity. The powerset inherits the space dimension of the polyhedron.

Parameters

ph The closed polyhedron to be used to build the powerset.

complexity The maximal complexity of any algorithms used.

Exceptions

std::length_error Thrown if the space dimension of `ph` exceeds the maximum allowed space dimension.

Definition at line 89 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::ANY_COMPLEXITY, Parma_Polyhedra_Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.9 template<typename PSET > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Grid & *gr*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of a grid.

If the grid is nonempty, builds a powerset containing a single disjunct approximating the grid. Builds the empty powerset otherwise. The powerset inherits the space dimension of the grid.

Parameters

gr The grid to be used to build the powerset.

complexity This argument is ignored.

Exceptions

std::length_error Thrown if the space dimension of `gr` exceeds the maximum allowed space dimension.

Definition at line 105 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Grid::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.10 template<typename PSET > template<typename T > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Octagonal_Shape< T > & *os*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of an octagonal shape.

If the octagonal shape is nonempty, builds a powerset containing a single disjunct approximating the octagonal shape. Builds the empty powerset otherwise. The powerset inherits the space dimension of the octagonal shape.

Parameters

os The octagonal shape to be used to build the powerset.

complexity This argument is ignored.

Exceptions

std::length_error Thrown if the space dimension of `os` exceeds the maximum allowed space dimension.

Definition at line 147 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Octagonal_Shape< T >::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.11 template<typename PSET > template<typename T > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const BD_Shape< T > & *bds*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of a bd shape.

If the bd shape is nonempty, builds a powerset containing a single disjunct approximating the bd shape. Builds the empty powerset otherwise. The powerset inherits the space dimension of the bd shape.

Parameters

bds The bd shape to be used to build the powerset.

complexity This argument is ignored.

Exceptions

std::length_error Thrown if the space dimension of `bdss` exceeds the maximum allowed space dimension.

Definition at line 158 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::BD_Shape< T >::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.12 template<typename PSET > template<typename Interval > Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset (const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, explicit]

Builds a pointset_powerset out of a box.

If the box is nonempty, builds a powerset containing a single disjunct approximating the box. Builds the empty powerset otherwise. The powerset inherits the space dimension of the box.

Parameters

box The box to be used to build the powerset.

complexity This argument is ignored.

Exceptions

std::length_error Thrown if the space dimension of `box` exceeds the maximum allowed space dimension.

Definition at line 136 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.13 template<> Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< QH > & *y*, Complexity_Class) [inline]

11.157.3.14 template<> Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< C_Polyhedron > & *y*, Complexity_Class) [inline]

11.157.3.15 template<> Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< Grid > & *y*, Complexity_Class) [inline]

11.157.3.16 template<> Parma_Polyhedra_Library::Pointset_Powerset< C_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< NNC_Polyhedron > & *y*, Complexity_Class) [inline]

11.157.3.17 template<> Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< QH > & *y*, Complexity_Class *complexity*) [inline]

Definition at line 61 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::reduced, Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.18 template<> Parma_Polyhedra_Library::Pointset_Powerset< PPL::NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< C_Polyhedron > & *y*, Complexity_Class) [inline]

Definition at line 297 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::reduced, Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.19 template<> Parma_Polyhedra_Library::Pointset_Powerset< PPL::NNC_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< Grid > & *y*, Complexity_Class) [inline]

Definition at line 312 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.3.20 template<> Parma_Polyhedra_Library::Pointset_Powerset< PPL::C_Polyhedron >::Pointset_Powerset (const Pointset_Powerset< NNC_Polyhedron > & *y*, Complexity_Class) [inline]

Definition at line 327 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4 Member Function Documentation

11.157.4.1 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruence (const Congruence & *c*) [inline]

Intersects `*this` with congruence `c`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `c` are topology-incompatible or dimension-incompatible.

Definition at line 184 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.2 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruences (const Congruence_System & *cgs*) [inline]

Intersects `*this` with the congruences in `cgs`.

Parameters

cgs The congruences to intersect with.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are topology-incompatible or dimension-incompatible.

Definition at line 206 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.3 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraint (const Constraint & *c*) [inline]

Intersects `*this` with constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are topology-incompatible or dimension-incompatible.

Definition at line 140 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.4 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraints (const Constraint_System & *cs*) [inline]

Intersects `*this` with the constraints in `cs`.

Parameters

cs The constraints to intersect with.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 162 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.5 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct (const PSET & *ph*) [inline]

Adds to `*this` the disjunct `ph`.

Exceptions

std::invalid_argument Thrown if `*this` and `ph` are dimension-incompatible.

Definition at line 44 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_load(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition_aux().

11.157.4.6 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_embed (dimension_type *m*) [inline]

Adds `m` new dimensions to the vector space containing `*this` and embeds each disjunct in `*this` in the new space.

Definition at line 252 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.7 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_project (dimension_type *m*) [inline]

Adds `m` new dimensions to the vector space containing `*this` without embedding the disjuncts in `*this` in the new space.

Definition at line 263 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.8 template<typename PSET > dimension_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 492 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Polyhedron::affine_dimension(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Polyhedron::poly_hull_assign(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.9 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the affine image of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 358 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.10 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) `[inline]`

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 376 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.11 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump (std::ostream & *s*) const `[inline]`

Writes to `s` an ASCII representation of `*this`.

Definition at line 1480 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::size(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.12 template<typename PSET> void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.157.4.13 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_load (std::istream & *s*) `[inline]`

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 1493 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.14 template<typename PSET > template<typename Widening > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_extrapolation_assign (const Pointset_Powerset< PSET > & *y*, Widening *wf*, unsigned *max_disjuncts*) [inline]

Assigns to `*this` the result of applying the BGP99 extrapolation operator to `*this` and `y`, using the widening function `wf` and the cardinality threshold `max_disjuncts`.

Parameters

y A powerset that *must* definitely entail `*this`;

wf The widening function to be used on polyhedra objects. It is obtained from the corresponding widening method by using the helper function Parma_Polyhedra_Library::widen_fun_ref. Legal values are, e.g., `widen_fun_ref(&Polyhedron::H79_widening_assign)` and `widen_fun_ref(&Polyhedron::limited_H79_extrapolation_assign, cs)`;

max_disjuncts The maximum number of disjuncts occurring in the powerset `*this` *before* starting the computation. If this number is exceeded, some of the disjuncts in `*this` are collapsed (i.e., joined together).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

For a description of the extrapolation operator, see [BGP99] and [BHZ03b].

Definition at line 1278 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::definitely_entails(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce().

11.157.4.15 template<typename PSET > template<typename Widening > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign (const Pointset_Powerset< PSET > & *y*, Widening *wf*) [inline, private]

Assigns to `*this` the result of applying the BGP99 heuristics to `*this` and `y`, using the widening function `wf`.

Definition at line 1230 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::definitely_entails(), Parma_Polyhedra_Library::EMPTY,

Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, Parma_Polyhedra_Library::Powerset< D >::size(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_extrapolation_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign().

11.157.4.16 template<typename PSET > template<typename Cert , typename Widening > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign (const Pointset_Powerset< PSET > & *y*, Widening *wf*) [inline]

Assigns to `*this` the result of computing the BHZ03-widening between `*this` and `y`, using the widening function `wf` certified by the convergence certificate `Cert`.

Parameters

y The finite powerset computed in the previous iteration step. It *must* definitely entail `*this`;

wf The widening function to be used on disjuncts. It is obtained from the corresponding widening method by using the helper function widen_fun_ref. Legal values are, e.g., `widen_fun_ref(&Polyhedron::H79_widening_assign)` and `widen_fun_ref(&Polyhedron::limited_H79_extrapolation_assign, cs)`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Warning

In order to obtain a proper widening operator, the template parameter `Cert` should be a finite convergence certificate for the base-level widening function `wf`; otherwise, an extrapolation operator is obtained. For a description of the methods that should be provided by `Cert`, see BHRZ03_Certificate or H79_Certificate.

Definition at line 1364 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::definitely_entails(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_cert_multiset_stabilizing(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), Parma_Polyhedra_Library::Powerset< D >::size(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.17 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 460 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.18 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`) [inline]

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 476 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.19 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from above in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 887 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.20 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from below in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 900 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.21 template<typename PSET > template<typename Cert > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates (std::map< Cert, size_type, typename Cert::Compare > & *cert_ms*) const [inline, private]

Records in `cert_ms` the certificates for this set of disjuncts.

Definition at line 1303 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), and Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_cert_multiset_stabilizing().

11.157.4.22 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Assigns to `*this` the concatenation of `*this` and `y`.

The result is obtained by computing the pairwise concatenation of each disjunct in `*this` with each disjunct in `y`.

Definition at line 103 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::abandon_expensive_computations, Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Determinate< PSET >::concatenate_assign(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Determinate< PSET >::is_bottom(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::sequence, Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.23 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains (Variable *var*) const [inline]

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Note

A variable is constrained if there exists a non-redundant disjunct that is constraining the variable: this definition relies on the powerset lattice structure and may be somewhat different from the geometric intuition. For instance, variable x is constrained in the powerset

$$ps = \{\{x \geq 0\}, \{x \leq 0\}\},$$

even though ps is geometrically equal to the whole vector space.

Definition at line 588 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension(), and Parma_Polyhedra_Library::Variable::space_dimension().

11.157.4.24 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::contains (const Pointset_Powerset< PSET > & *y*) const [inline]

Returns `true` if and only if each disjunct of `y` is contained in a disjunct of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 737 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.25 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::contains_integer_point () const [inline]

Returns `true` if and only if `*this` contains at least one integer point.

Definition at line 1148 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.26 template<> void Parma_Polyhedra_Library::Pointset_Powerset< PPL::Grid >::difference_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Definition at line 258 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.27 template<> void Parma_Polyhedra_Library::Pointset_Powerset< PPL::NNC_Polyhedron >::difference_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Definition at line 34 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.28 template<> void Parma_Polyhedra_Library::Pointset_Powerset< Grid >::difference_assign (const Pointset_Powerset< PSET > & *y*) [inline]

11.157.4.29 template<> void Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::difference_assign (const Pointset_Powerset< PSET > & *y*) [inline]

11.157.4.30 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Assigns to `*this` an (a smallest) over-approximation as a powerset of the disjunct domain of the set-theoretical difference of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 283 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign().

11.157.4.31 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 630 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.32 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 643 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.33 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::expand_space_dimension (Variable *var*, dimension_type *m*) [inline]

Creates m copies of the space dimension corresponding to var.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if var does not correspond to a dimension of the vector space.

std::length_error Thrown if adding m new space dimensions would cause the vector space to exceed dimension max_space_dimension().

If *this has space dimension n, with $n > 0$, and var has space dimension $k \leq n$, then the k-th space dimension is expanded to m new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 331 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.34 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by *this.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 264 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::external_memory_in_bytes().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::total_memory_in_bytes().

11.157.4.35 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*) [inline]

Folds the space dimensions in vars into dest.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 343 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.36 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\text{lhs}' \bowtie \text{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 396 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.37 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\text{var}' \bowtie \frac{\text{expr}}{\text{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 426 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.38 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 411 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.39 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one()) [inline]

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 442 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.40 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< PPL::Grid >::geometrically_covers (const Pointset_Powerset< PSET > & *y*) const [inline]

Definition at line 286 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), and Parma_Polyhedra_Library::Powerset< D >::end().

11.157.4.41 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< PPL::NNC_Polyhedron >::geometrically_covers (const Pointset_Powerset< PSET > & *y*) const [inline]

Definition at line 61 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), and Parma_Polyhedra_Library::Powerset< D >::end().

11.157.4.42 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< Grid >::geometrically_covers (const Pointset_Powerset< PSET > & *y*) const [inline]

11.157.4.43 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::geometrically_covers (const Pointset_Powerset< PSET > & *y*) const [inline]

11.157.4.44 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers (const Pointset_Powerset< PSET > & *y*) const [inline]

Returns `true` if and only if `*this` geometrically covers `y`, i.e., if any point (in some element) of `y` is also a point (of some element) of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Warning

This may be *really* expensive!

Definition at line 229 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_equals().

11.157.4.45 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< NNC_Polyhedron >::geometrically_equals (const Pointset_Powerset< PSET > & *y*) const [inline]

Definition at line 257 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers().

11.157.4.46 template<> bool Parma_Polyhedra_Library::Pointset_Powerset< Grid >::geometrically_equals (const Pointset_Powerset< PSET > & *y*) const [inline]

Definition at line 249 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers().

11.157.4.47 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_equals (const Pointset_Powerset< PSET > & *y*) const [inline]

Returns `true` if and only if `*this` is geometrically equal to `y`, i.e., if (the elements of) `*this` and `y` contain the same set of points.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Warning

This may be *really* expensive!

Definition at line 239 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers().

11.157.4.48 template<typename PSET > int32_t Parma_Polyhedra_Library::Pointset_Powerset< PSET >::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 276 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension().

11.157.4.49 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Assigns to `*this` the intersection of `*this` and `y`.

The result is obtained by intersecting each disjunct in `*this` with each disjunct in `y` and collecting all these intersections.

Definition at line 210 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::lift_op_assign(), and Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign().

11.157.4.50 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_preserving_enlarge_element (PSET & *dest*) const [inline, private]

Assigns to `dest` a powerset meet-preserving enlargement of itself with respect to `*this`. If `false` is returned, then the intersection is empty.

Note

It is assumed that `*this` and `dest` are topology-compatible and dimension-compatible.

Definition at line 665 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence, Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension(), and Parma_Polyhedra_Library::UNIVERSE.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign().

11.157.4.51 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_bounded () const [inline]

Returns `true` if and only if all elements in `*this` are bounded.

Definition at line 577 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.52 template<typename PSET > template<typename Cert > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_cert_multiset_stabilizing (const std::map< Cert, size_type, typename Cert::Compare > & *y_cert_ms*) const [inline, private]

Returns `true` if and only if the current set of dijsuncts is stabilizing with respect to the multiset of certificates `y_cert_ms`.

Definition at line 1318 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign().

11.157.4.53 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_discrete () const [inline]

Returns `true` if and only if `*this` is discrete.

Definition at line 552 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.54 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_disjoint_from (const Pointset_Powerset< PSET > & *y*) const [inline]

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 612 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.55 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty () const [inline]

Returns `true` if and only if `*this` is an empty powerset.

Definition at line 541 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign().

11.157.4.56 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_topologically_closed () const [inline]

Returns `true` if and only if all the disjuncts in `*this` are topologically closed.

Definition at line 563 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.57 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe () const [inline]

Returns `true` if and only if `*this` is the top element of the powerser lattice.

Definition at line 519 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced(), Parma_Polyhedra_Library::Powerset< D >::size(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap(), and Parma_Polyhedra_Library::UNIVERSE.

11.157.4.58 template<typename PSET > template<typename Partial_Function > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions (const Partial_Function & *pfunc*) [inline]

Remaps the dimensions of the vector space according to a partial function.

See also Polyhedron::map_space_dimensions.

Definition at line 307 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::is_bottom(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.59 template<typename PSET > dimension_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::max_space_dimension () [inline, static]

Returns the maximum space dimension a Pointset_Powerset<PSET> can handle.

Definition at line 48 of file Pointset_Powerset.inlines.hh.

11.157.4.60 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value;

g When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d`, `maximum` and `g` are left untouched.

Definition at line 967 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.61 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

Definition at line 912 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.62 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value;

g When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

Definition at line 1085 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.63 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 1030 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.64 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK () const [inline]

Checks if all the invariants are satisfied.

Definition at line 1527 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraints(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_load(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::expand_space_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::fold_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraints(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::topological_closure_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::unconstrain(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::wrap_assign().

11.157.4.65 template<typename PSET > template<typename QH > Pointset_Powerset< PSET > & Parma_Polyhedra_Library::Pointset_Powerset< PSET >::operator= (const Pointset_Powerset< QH > & *y*) [inline]

Conversion assignment: the type `QH` of the disjuncts in the source powerset is different from `PSET` (`*this` and `y` can be dimension-incompatible).

Definition at line 201 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.66 template<typename PSET > Pointset_Powerset< PSET > & Parma_Polyhedra_Library::Pointset_Powerset< PSET >::operator= (const Pointset_Powerset< PSET > & *y*) [inline]

The assignment operator (`*this` and `y` can be dimension-incompatible).

Definition at line 183 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.67 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce () [inline]

Assign to `*this` the result of (recursively) merging together the pairs of disjuncts whose upper-bound is the same as their set-theoretical union.

On exit, for all the pairs $\mathcal{P}$, $\mathcal{Q}$ of different disjuncts in `*this`, we have $\mathcal{P} \uplus \mathcal{Q} \neq \mathcal{P} \cup \mathcal{Q}$.

Definition at line 1177 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::sequence, Parma_Polyhedra_Library::Powerset< D >::size(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_extrapolation_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign().

11.157.4.68 template<typename PSET> void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.157.4.69 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruence (const Congruence & *cg*) [inline]

Use the congruence `cg` to refine `*this`.

Parameters

cg The congruence to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cg` are dimension-incompatible.

Definition at line 195 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.70 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruences (const Congruence_System & *cgs*) [inline]

Use the congruences in `cgs` to refine `*this`.

Parameters

cgs The congruences to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 217 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.71 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraint (const Constraint & *c*) [inline]

Use the constraint `c` to refine `*this`.

Parameters

c The constraint to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `c` are dimension-incompatible.

Definition at line 151 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.72 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraints (const Constraint_System & *cs*) [inline]

Use the constraints in `cs` to refine `*this`.

Parameters

cs The constraints to be used for refinement.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 173 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.73 template<typename PSET > Poly_Con_Relation Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with (const Congruence & *cg*) const [inline]

Returns the relations holding between the powerset `*this` and the congruence `c`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `c` are dimension-incompatible.

Definition at line 783 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Poly_Con_Relation::nothing(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects().

11.157.4.74 template<typename PSET > Poly_Gen_Relation Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with (const Generator & *g*) const [inline]

Returns the relations holding between the powerset `*this` and the generator `g`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible.

Definition at line 871 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Poly_Gen_Relation::implies(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes().

11.157.4.75 template<typename PSET > Poly_Con_Relation Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with (const Constraint & *c*) const [inline]

Returns the relations holding between the powerset `*this` and the constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 827 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Poly_Con_Relation::nothing(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects().

11.157.4.76 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_higher_space_dimensions (dimension_type *new_dimension*) [inline]

Removes the higher space dimensions so that the resulting space will have dimension `new_dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimensions` is greater than the space dimension of `*this`.

Definition at line 291 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.77 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_space_dimensions (const Variables_Set & *vars*) [inline]

Removes all the specified space dimensions.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 274 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

11.157.4.78 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign (const Pointset_Powerset< PSET > & y) [inline]

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 689 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_preserving_enlarge_element(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Powerset< D >::size().

11.157.4.79 template<typename PSET > dimension_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 42 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::hash_code(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_preserving_enlarge_element(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe().

11.157.4.80 template<typename PSET > bool Parma_Polyhedra_Library::Pointset_Powerset< PSET >::strictly_contains (const Pointset_Powerset< PSET > & y) const [inline]

Returns `true` if and only if each disjunct of `y` is strictly contained in a disjunct of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 758 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.81 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap (Pointset_Powerset< PSET > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 192 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_load(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::operator=(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.4.82 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::time_elapse_assign (const Pointset_Powerset< PSET > & *y*) [inline]

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

The result is obtained by computing the pairwise time elapse of each disjunct in `*this` with each disjunct in `y`.

Definition at line 219 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::lift_op_assign(), and Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign().

11.157.4.83 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::topological_closure_assign () [inline]

Assigns to `*this` its topological closure.

Definition at line 654 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.84 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented from Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >.

Definition at line 270 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::external_memory_in_bytes().

11.157.4.85 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::unconstrain (const Variables_Set & *vars*) [inline]

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 240 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.86 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::unconstrain (Variable *var*) [inline]

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 228 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.4.87 template<typename PSET > void Parma_Polyhedra_Library::Pointset_Powerset< PSET >::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true) [inline]

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system whose variables are contained in `vars`. If `*pcs` depends on variables not in `vars`, the behavior is undefined. When non-null, the pointed-to constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in `*pcs`.

complexity_threshold A precision parameter of the wrapping operator: higher values result in possibly improved precision.

wrap_individually `true` if the dimensions should be wrapped individually (something that results in much greater efficiency to the detriment of precision).

Exceptions

std::invalid_argument Thrown if `*pcs` is dimension-incompatible with `vars`, or if `*this` is dimension-incompatible `vars` or with `*pcs`.

Definition at line 1159 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.157.5 Friends And Related Function Documentation

11.157.5.1 template<typename PSET> std::pair< PPL::Grid, PPL::Pointset_Powerset< PPL::Grid > > approximate_partition (const Grid & *p*, const Grid & *q*, bool & *finite_partition*) [related]

Partitions the grid `q` with respect to grid `p` if and only if such a partition is finite.

Let `p` and `q` be two grids. The function returns an object `r` of type `std::pair<PSET, Pointset_Powerset<Grid> >` such that

- `r.first` is the intersection of `p` and `q`;
- If there is a finite partition of `q` wrt `p` the Boolean `finite_partition` is set to true and `r.second` has the property that all its elements are pairwise disjoint and disjoint from `p` and the set-theoretical union of `r.first` with all the elements of `r.second` gives `q` (i.e., `r` is the representation of a partition of `q`).
- Otherwise the Boolean `finite_partition` is set to false and the singleton set that contains `q` is stored in `r.second`r.

Definition at line 182 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(), Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Grid::congruences(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Grid::minimized_congruences(), and Parma_Polyhedra_Library::Grid::space_dimension().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign().

11.157.5.2 template<typename PSET> bool approximate_partition_aux (const PPL::Congruence & *c*, PPL::Grid & *qq*, PPL::Pointset_Powerset< PPL::Grid > & *r*) [related]

Uses the congruence `c` to approximately partition the grid `qq`.

On exit, the intersection of `qq` and congruence `c` is stored in `qq`, whereas a finite set of grids whose set-theoretic union contains the intersection of `qq` with the negation of `c` is added, as a set of new disjuncts, to the powerset `r`.

Definition at line 124 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Grid::add_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Grid::congruences(), Parma_Polyhedra_Library::Congruence::inhomogeneous_term(), Parma_Polyhedra_Library::Grid::is_empty(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Congruence::modulus(), Parma_Polyhedra_Library::Congruence_System::num_equalities(), Parma_Polyhedra_Library::Congruence_System::num_proper_congruences(), PPL_DIRTY_TEMP_COEFFICIENT, and Parma_Polyhedra_Library::rem_assign().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition().

11.157.5.3 bool check_containment (const C_Polyhedron & *ph*, const Pointset_Powerset< C_Polyhedron > & *ps*) [related]

Definition at line 304 of file Pointset_Powerset.inlines.hh.

11.157.5.4 template<typename PSET > bool check_containment (const PSET & *ph*, const Pointset_Powerset< PSET > & *ps*) [related]

Returns `true` if and only if the union of the objects in `ps` contains `ph`.

Note

It is assumed that the template parameter PSET can be converted without precision loss into an NNC_Polyhedron; otherwise, an incorrect result might be obtained.

Definition at line 294 of file Pointset_Powerset.inlines.hh.

11.157.5.5 template<typename PSET> bool check_containment (const Grid & *ph*, const Pointset_Powerset< Grid > & *ps*) [related]

Returns `true` if and only if the union of the grids `ps` contains the grid `g`.

Definition at line 204 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition(), Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Grid::contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Grid::is_disjoint_from(), Parma_Polyhedra_Library::Grid::is_empty(), and Parma_Polyhedra_Library::Grid::space_dimension().

11.157.5.6 template<typename PSET> bool check_containment (const NNC_Polyhedron & *ph*, const Pointset_Powerset< NNC_Polyhedron > & *ps*) [related]

Returns `true` if and only if the union of the NNC polyhedra in `ps` contains the NNC polyhedron `ph`.

Definition at line 71 of file Pointset_Powerset.cc.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), Parma_Polyhedra_Library::Polyhedron::contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end(), Parma_Polyhedra_Library::Polyhedron::is_disjoint_from(), Parma_Polyhedra_Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition(), and Parma_Polyhedra_Library::Polyhedron::space_dimension().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers().

11.157.5.7 template<typename PSET > std::pair< PSET, Pointset_Powerset< NNC_Polyhedron > > linear_partition (const PSET & *p*, const PSET & *q*) [related]

Partitions `q` with respect to `p`.

Let `p` and `q` be two polyhedra. The function returns an object `r` of type `std::pair<PSET, Pointset_Powerset<NNC_Polyhedron> >` such that

- `r.first` is the intersection of `p` and `q`;
- `r.second` has the property that all its elements are pairwise disjoint and disjoint from `p`;
- the set-theoretical union of `r.first` with all the elements of `r.second` gives `q` (i.e., `r` is the representation of a partition of `q`).

See `this paper` for more information about the implementation.

Definition at line 1578 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::is_equality(), and Parma_Polyhedra_Library::Checked::le.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign().

11.157.5.8 template<typename PSET > void linear_partition_aux (const Constraint & *c*, PSET & *qq*, Pointset_Powerset< NNC_Polyhedron > & *r*) [related]

Partitions polyhedron `qq` according to constraint `c`.

On exit, the intersection of `qq` and constraint `c` is stored in `qq`, whereas the intersection of `qq` with the negation of `c` is added as a new disjunct of the powerset `r`.

Definition at line 1558 of file Pointset_Powerset.templates.hh.

References Parma_Polyhedra_Library::Polyhedron::add_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Polyhedron::is_empty(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), and Parma_Polyhedra_Library::Checked::le.

11.157.5.9 template<typename PSET> friend class Pointset_Powerset< NNC_Polyhedron > [friend]

Definition at line 1302 of file Pointset_Powerset.defs.hh.

11.157.5.10 template<typename PSET > void swap (Parma_Polyhedra_Library::Pointset_Powerset< PSET > & *x*, Parma_Polyhedra_Library::Pointset_Powerset< PSET > & *y*) [related]

Specializes `std::swap`.

Definition at line 318 of file Pointset_Powerset.inlines.hh.

References Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

11.157.5.11 template<typename PSET , typename CSYS > Limited_Widening_Function< PSET, CSYS > widen_fun_ref (void(PSET::*)(const PSET &, const CSYS &, unsigned *) *lwm*, const CSYS & *cs*) [related]

Wraps a limited widening method into a function object.

Parameters

lwm The limited widening method.

cs The constraint system limiting the widening.

Definition at line 67 of file Widening_Function.inlines.hh.

11.157.5.12 template<typename PSET > Widening_Function< PSET > widen_fun_ref (void(PSET::*)(const PSET &, unsigned *) *wm*) [related]

Wraps a widening method into a function object.

Parameters

wm The widening method.

Definition at line 60 of file Widening_Function.inlines.hh.

11.157.6 Member Data Documentation

11.157.6.1 template<typename PSET> dimension_type Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dim [private]

The number of dimensions of the enclosing vector space.

Definition at line 1264 of file Pointset_Powerset.defs.hh.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_load(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::expand_space_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::fold_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::operator=(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::space_dimension(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::swap().

The documentation for this class was generated from the following files:

- Pointset_Powerset.defs.hh
- Widening_Function.defs.hh
- Widening_Function.inlines.hh
- Pointset_Powerset.inlines.hh
- Pointset_Powerset.templates.hh
- Pointset_Powerset.cc

11.158 Parma_Polyhedra_Library::Poly_Con_Relation Class Reference

The relation between a polyhedron and a constraint.

`#include <Poly_Con_Relation.defs.hh>`

Public Member Functions

- flags_t get_flags () const

  Access the internal flags: this is needed for some language interfaces.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool implies (const Poly_Con_Relation &y) const

  True if and only if `*this` *implies* `y`.

- bool OK () const

  Checks if all the invariants are satisfied.

Static Public Member Functions

- static Poly_Con_Relation nothing ()

  The assertion that says nothing.

- static Poly_Con_Relation is_disjoint ()

  The polyhedron and the set of points satisfying the constraint are disjoint.

- static Poly_Con_Relation strictly_intersects ()

  The polyhedron intersects the set of points satisfying the constraint, but it is not included in it.

- static Poly_Con_Relation is_included ()

  The polyhedron is included in the set of points satisfying the constraint.

- static Poly_Con_Relation saturates ()

  The polyhedron is included in the set of points saturating the constraint.

Private Types

- typedef unsigned int flags_t

  Poly_Con_Relation is implemented by means of a finite bitset.

Private Member Functions

- Poly_Con_Relation (flags_t mask)

  Construct from a bit-mask.

Static Private Member Functions

- static bool implies (flags_t x, flags_t y)

  True if and only if the conjunction `x` *implies the conjunction* `y`.

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

- static const flags_t EVERYTHING

  All assertions together.

Bit-masks for the individual assertions

- static const flags_t NOTHING = 0U
- static const flags_t IS_DISJOINT = 1U << 0
- static const flags_t STRICTLY_INTERSECTS = 1U << 1
- static const flags_t IS_INCLUDED = 1U << 2
- static const flags_t SATURATES = 1U << 3

Friends

- bool operator== (const Poly_Con_Relation &x, const Poly_Con_Relation &y)

  True if and only if `x` *and* `y` *are logically equivalent.*

- bool operator!= (const Poly_Con_Relation &x, const Poly_Con_Relation &y)

  True if and only if `x` *and* `y` *are not logically equivalent.*

- Poly_Con_Relation operator&& (const Poly_Con_Relation &x, const Poly_Con_Relation &y)

  Yields the logical conjunction of `x` *and* `y`.

- Poly_Con_Relation operator- (const Poly_Con_Relation &x, const Poly_Con_Relation &y)

  Yields the assertion with all the conjuncts of `x` *that are not in* `y`.

- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Poly_Con_Relation &r)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Poly_Con_Relation &r)

  Output operator.

11.158.1 Detailed Description

The relation between a polyhedron and a constraint. This class implements conjunctions of assertions on the relation between a polyhedron and a constraint.

Definition at line 73 of file Poly_Con_Relation.defs.hh.

11.158.2 Member Typedef Documentation

11.158.2.1 typedef unsigned int Parma_Polyhedra_Library::Poly_Con_Relation::flags_t [private]

Poly_Con_Relation is implemented by means of a finite bitset.

Definition at line 76 of file Poly_Con_Relation.defs.hh.

11.158.3 Constructor & Destructor Documentation

11.158.3.1 Parma_Polyhedra_Library::Poly_Con_Relation::Poly_Con_Relation (flags_t *mask*) [inline, private]

Construct from a bit-mask.

Definition at line 30 of file Poly_Con_Relation.inlines.hh.

Referenced by is_disjoint(), is_included(), nothing(), saturates(), and strictly_intersects().

11.158.4 Member Function Documentation

11.158.4.1 void Parma_Polyhedra_Library::Poly_Con_Relation::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 35 of file Poly_Con_Relation.cc.

References flags, implies(), IS_DISJOINT, IS_INCLUDED, NOTHING, SATURATES, and STRICTLY_-INTERSECTS.

11.158.4.2 void Parma_Polyhedra_Library::Poly_Con_Relation::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.158.4.3 Poly_Con_Relation::flags_t Parma_Polyhedra_Library::Poly_Con_Relation::get_flags () const [inline]

Access the internal flags: this is needed for some language interfaces.

Definition at line 35 of file Poly_Con_Relation.inlines.hh.

References flags.

11.158.4.4 bool Parma_Polyhedra_Library::Poly_Con_Relation::implies (const Poly_Con_Relation & *y*) const [inline]

True if and only if `*this` implies `y`.

Definition at line 70 of file Poly_Con_Relation.inlines.hh.

References flags, and implies().

11.158.4.5 bool Parma_Polyhedra_Library::Poly_Con_Relation::implies (flags_t *x*, flags_t *y*) [inline, static, private]

True if and only if the conjunction `x` implies the conjunction `y`.

Definition at line 65 of file Poly_Con_Relation.inlines.hh.

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Grid::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), implies(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.158.4.6 Poly_Con_Relation Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint () [inline, static]

The polyhedron and the set of points satisfying the constraint are disjoint.

Definition at line 45 of file Poly_Con_Relation.inlines.hh.

References IS_DISJOINT, and Poly_Con_Relation().

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.158.4.7 Poly_Con_Relation Parma_Polyhedra_Library::Poly_Con_Relation::is_included () [inline, static]

The polyhedron is included in the set of points satisfying the constraint.

Definition at line 55 of file Poly_Con_Relation.inlines.hh.

References IS_INCLUDED, and Poly_Con_Relation().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Octagonal_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Grid::difference_assign(), Parma_Polyhedra_Library::BD_Shape< T >::difference_assign(), Parma_Polyhedra_Library::Box< ITV >::get_limiting_box(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Grid::limited_congruence_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_extrapolation_assign(), Parma_Polyhedra_Library::Grid::limited_generator_extrapolation_assign(), Parma_Polyhedra_Library::Polyhedron::poly_difference_assign(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::shrink_to_congruence_no_check(), and Parma_Polyhedra_Library::Grid::simplify_using_context_assign().

11.158.4.8 Poly_Con_Relation Parma_Polyhedra_Library::Poly_Con_Relation::nothing () [inline, static]

The assertion that says nothing.

Definition at line 40 of file Poly_Con_Relation.inlines.hh.

References NOTHING, and Poly_Con_Relation().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), and Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with().

11.158.4.9 bool Parma_Polyhedra_Library::Poly_Con_Relation::OK () const

Checks if all the invariants are satisfied.

Definition at line 76 of file Poly_Con_Relation.cc.

11.158.4.10 void Parma_Polyhedra_Library::Poly_Con_Relation::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.158.4.11 Poly_Con_Relation Parma_Polyhedra_Library::Poly_Con_Relation::saturates () [inline, static]

The polyhedron is included in the set of points saturating the constraint.

Definition at line 60 of file Poly_Con_Relation.inlines.hh.

References Poly_Con_Relation(), and SATURATES.

Referenced by Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.158.4.12 Poly_Con_Relation Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects () [inline, static]

The polyhedron intersects the set of points satisfying the constraint, but it is not included in it.

Definition at line 50 of file Poly_Con_Relation.inlines.hh.

References Poly_Con_Relation(), and STRICTLY_INTERSECTS.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::interval_relation(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.158.5 Friends And Related Function Documentation

11.158.5.1 bool operator!= (const Poly_Con_Relation & *x*, const Poly_Con_Relation & *y*) [friend]

True if and only if x and y are not logically equivalent.

Definition at line 82 of file Poly_Con_Relation.inlines.hh.

11.158.5.2 Poly_Con_Relation operator&& (const Poly_Con_Relation & *x*, const Poly_Con_Relation & *y*) [friend]

Yields the logical conjunction of x and y.

Definition at line 88 of file Poly_Con_Relation.inlines.hh.

11.158.5.3 Poly_Con_Relation operator- (const Poly_Con_Relation & *x*, const Poly_Con_Relation & *y*) [friend]

Yields the assertion with all the conjuncts of x that are not in y.

Definition at line 94 of file Poly_Con_Relation.inlines.hh.

11.158.5.4 std::ostream & operator<< (std::ostream & *s*, const Poly_Con_Relation & *r*) [related]

Output operator.

Definition at line 70 of file Poly_Con_Relation.cc.

11.158.5.5 bool operator== (const Poly_Con_Relation & *x*, const Poly_Con_Relation & *y*) [friend]

True if and only if x and y are logically equivalent.

Definition at line 76 of file Poly_Con_Relation.inlines.hh.

11.158.5.6 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Poly_Con_Relation & *r*) [friend]

11.158.6 Member Data Documentation

11.158.6.1 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::EVERYTHING [static, private]

Initial value:

```
IS_DISJOINT
  | STRICTLY_INTERSECTS
  | IS_INCLUDED
  | SATURATES
```

All assertions together.

Definition at line 89 of file Poly_Con_Relation.defs.hh.

11.158.6.2 flags_t Parma_Polyhedra_Library::Poly_Con_Relation::flags [private]

This holds the current bitset.

Definition at line 95 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), get_flags(), and implies().

11.158.6.3 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::IS_DISJOINT = 1U << 0 [static, private]

Definition at line 81 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), and is_disjoint().

11.158.6.4 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::IS_INCLUDED = 1U << 2 [static, private]

Definition at line 83 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), and is_included().

11.158.6.5 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::NOTHING = 0U [static, private]

Definition at line 80 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), and nothing().

11.158.6.6 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::SATURATES = 1U << 3 [static, private]

Definition at line 84 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), and saturates().

11.158.6.7 const flags_t Parma_Polyhedra_Library::Poly_Con_Relation::STRICTLY_INTERSECTS = 1U << 1 [static, private]

Definition at line 82 of file Poly_Con_Relation.defs.hh.

Referenced by ascii_dump(), and strictly_intersects().

The documentation for this class was generated from the following files:

- Poly_Con_Relation.defs.hh
- Poly_Con_Relation.inlines.hh
- Poly_Con_Relation.cc

11.159 Parma_Polyhedra_Library::Poly_Gen_Relation Class Reference

The relation between a polyhedron and a generator.

`#include <Poly_Gen_Relation.defs.hh>`

Public Member Functions

- flags_t get_flags () const

  Access the internal flags: this is needed for some language interfaces.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool implies (const Poly_Gen_Relation &y) const

  True if and only if `*this` *implies* `y`.

- bool OK () const

  Checks if all the invariants are satisfied.

Static Public Member Functions

- static Poly_Gen_Relation nothing ()

  The assertion that says nothing.

- static Poly_Gen_Relation subsumes ()

  Adding the generator would not change the polyhedron.

Private Types

- typedef unsigned int flags_t

  Poly_Gen_Relation is implemented by means of a finite bitset.

Private Member Functions

- Poly_Gen_Relation (flags_t mask)

  Construct from a bit-mask.

Static Private Member Functions

- static bool implies (flags_t x, flags_t y)

  True if and only if the conjunction `x` *implies the conjunction* `y`.

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

- static const flags_t EVERYTHING = SUBSUMES

  All assertions together.

Bit-masks for the individual assertions

- static const flags_t NOTHING = 0U
- static const flags_t SUBSUMES = 1U << 0

Friends

- bool operator== (const Poly_Gen_Relation &x, const Poly_Gen_Relation &y)

  True if and only if `x` *and* `y` *are logically equivalent.*

- bool operator!= (const Poly_Gen_Relation &x, const Poly_Gen_Relation &y)

  True if and only if `x` *and* `y` *are not logically equivalent.*

- Poly_Gen_Relation operator&& (const Poly_Gen_Relation &x, const Poly_Gen_Relation &y)

  Yields the logical conjunction of `x` *and* `y`.

- Poly_Gen_Relation operator- (const Poly_Gen_Relation &x, const Poly_Gen_Relation &y)

  Yields the assertion with all the conjuncts of `x` *that are not in* `y`.

- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Poly_Gen_Relation &r)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Poly_Gen_Relation &r)

  Output operator.

11.159.1 Detailed Description

The relation between a polyhedron and a generator. This class implements conjunctions of assertions on the relation between a polyhedron and a generator.

Definition at line 73 of file Poly_Gen_Relation.defs.hh.

11.159.2 Member Typedef Documentation

11.159.2.1 typedef unsigned int Parma_Polyhedra_Library::Poly_Gen_Relation::flags_t [private]

Poly_Gen_Relation is implemented by means of a finite bitset.

Definition at line 76 of file Poly_Gen_Relation.defs.hh.

11.159.3 Constructor & Destructor Documentation

11.159.3.1 Parma_Polyhedra_Library::Poly_Gen_Relation::Poly_Gen_Relation (flags_t *mask*) [inline, private]

Construct from a bit-mask.

Definition at line 30 of file Poly_Gen_Relation.inlines.hh.

Referenced by nothing(), and subsumes().

11.159.4 Member Function Documentation

11.159.4.1 void Parma_Polyhedra_Library::Poly_Gen_Relation::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 35 of file Poly_Gen_Relation.cc.

References flags, implies(), NOTHING, and SUBSUMES.

11.159.4.2 void Parma_Polyhedra_Library::Poly_Gen_Relation::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.159.4.3 Poly_Gen_Relation::flags_t Parma_Polyhedra_Library::Poly_Gen_Relation::get_flags () const [inline]

Access the internal flags: this is needed for some language interfaces.

Definition at line 35 of file Poly_Gen_Relation.inlines.hh.

References flags.

11.159.4.4 bool Parma_Polyhedra_Library::Poly_Gen_Relation::implies (const Poly_Gen_Relation & *y*) const [inline]

True if and only if `*this` implies `y`.

Definition at line 55 of file Poly_Gen_Relation.inlines.hh.

References flags, and implies().

11.159.4.5 bool Parma_Polyhedra_Library::Poly_Gen_Relation::implies (flags_t *x*, flags_t *y*) [inline, static, private]

True if and only if the conjunction `x` implies the conjunction `y`.

Definition at line 50 of file Poly_Gen_Relation.inlines.hh.

Referenced by ascii_dump(), implies(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with().

11.159.4.6 Poly_Gen_Relation Parma_Polyhedra_Library::Poly_Gen_Relation::nothing () [inline, static]

The assertion that says nothing.

Definition at line 40 of file Poly_Gen_Relation.inlines.hh.

References NOTHING, and Poly_Gen_Relation().

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.159.4.7 bool Parma_Polyhedra_Library::Poly_Gen_Relation::OK () const

Checks if all the invariants are satisfied.

Definition at line 64 of file Poly_Gen_Relation.cc.

11.159.4.8 void Parma_Polyhedra_Library::Poly_Gen_Relation::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.159.4.9 Poly_Gen_Relation Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes () [inline, static]

Adding the generator would not change the polyhedron.

Definition at line 45 of file Poly_Gen_Relation.inlines.hh.

References Poly_Gen_Relation(), and SUBSUMES.

Referenced by Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::relation_with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), and Parma_Polyhedra_Library::BD_Shape< T >::relation_with().

11.159.5 Friends And Related Function Documentation

11.159.5.1 bool operator!= (const Poly_Gen_Relation & *x*, const Poly_Gen_Relation & *y*) [friend]

True if and only if x and y are not logically equivalent.

Definition at line 67 of file Poly_Gen_Relation.inlines.hh.

11.159.5.2 Poly_Gen_Relation operator&& (const Poly_Gen_Relation & *x*, const Poly_Gen_Relation & *y*) [friend]

Yields the logical conjunction of x and y.

Definition at line 73 of file Poly_Gen_Relation.inlines.hh.

11.159.5.3 Poly_Gen_Relation operator- (const Poly_Gen_Relation & *x*, const Poly_Gen_Relation & *y*) [friend]

Yields the assertion with all the conjuncts of x that are not in y.

Definition at line 79 of file Poly_Gen_Relation.inlines.hh.

11.159.5.4 std::ostream & operator<< (std::ostream & *s*, const Poly_Gen_Relation & *r*) [related]

Output operator.

Definition at line 58 of file Poly_Gen_Relation.cc.

11.159.5.5 bool operator== (const Poly_Gen_Relation & *x*, const Poly_Gen_Relation & *y*) [friend]

True if and only if x and y are logically equivalent.

Definition at line 61 of file Poly_Gen_Relation.inlines.hh.

11.159.5.6 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Poly_Gen_Relation & *r*) [friend]

11.159.6 Member Data Documentation

11.159.6.1 const flags_t Parma_Polyhedra_Library::Poly_Gen_Relation::EVERYTHING = SUBSUMES [static, private]

All assertions together.

Definition at line 86 of file Poly_Gen_Relation.defs.hh.

11.159.6.2 flags_t Parma_Polyhedra_Library::Poly_Gen_Relation::flags [private]

This holds the current bitset.

Definition at line 89 of file Poly_Gen_Relation.defs.hh.

Referenced by ascii_dump(), get_flags(), and implies().

11.159.6.3 const flags_t Parma_Polyhedra_Library::Poly_Gen_Relation::NOTHING = 0U [static, private]

Definition at line 80 of file Poly_Gen_Relation.defs.hh.

Referenced by ascii_dump(), and nothing().

11.159.6.4 const flags_t Parma_Polyhedra_Library::Poly_Gen_Relation::SUBSUMES = 1U << 0 [static, private]

Definition at line 81 of file Poly_Gen_Relation.defs.hh.

Referenced by ascii_dump(), and subsumes().

The documentation for this class was generated from the following files:

- Poly_Gen_Relation.defs.hh

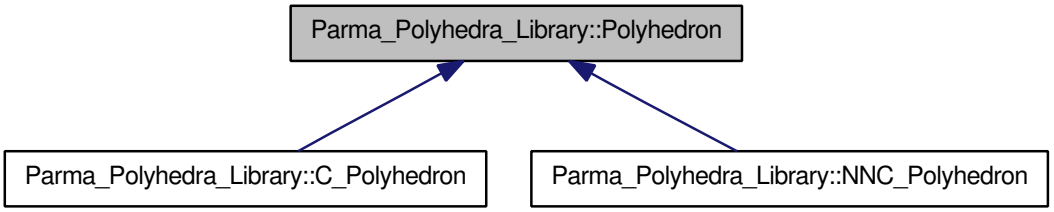
- Poly_Gen_Relation.inlines.hh
- Poly_Gen_Relation.cc

11.160 Parma_Polyhedra_Library::Polyhedron Class Reference

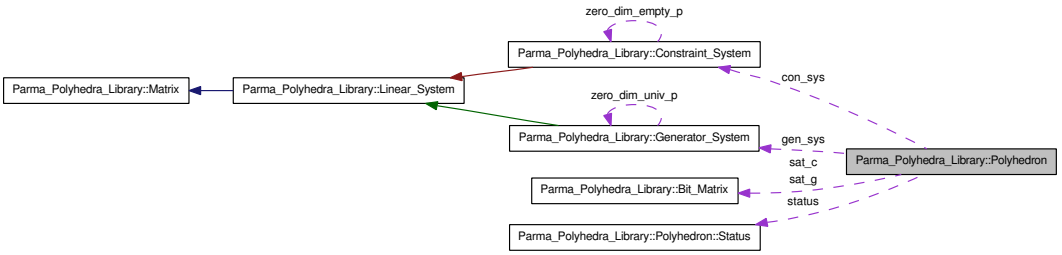
The base class for convex polyhedra.

`#include <Polyhedron.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Polyhedron:



Collaboration diagram for Parma_Polyhedra_Library::Polyhedron:



Classes

- class Status

  A conjunctive assertion about a polyhedron.

Public Types

- typedef Coefficient coefficient_type

  The numeric type of coefficients.

Public Member Functions

Member Functions that Do Not Modify the Polyhedron

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- dimension_type affine_dimension () const
  Returns 0, *if* `*this` *is empty; otherwise, returns the affine dimension of* `*this`.

- const Constraint_System & constraints () const
  Returns the system of constraints.

- const Constraint_System & minimized_constraints () const
  Returns the system of constraints, with no redundant constraint.

- const Generator_System & generators () const
  Returns the system of generators.

- const Generator_System & minimized_generators () const
  Returns the system of generators, with no redundant generator.

- Congruence_System congruences () const
  Returns a system of (equality) congruences satisfied by `*this`.

- Congruence_System minimized_congruences () const
  Returns a system of (equality) congruences satisfied by `*this`, *with no redundant congruences and having the same affine dimension as* `*this`.

- Grid_Generator_System grid_generators () const
  Returns a universe system of grid generators.

- Grid_Generator_System minimized_grid_generators () const
  Returns a universe system of grid generators.

- Poly_Con_Relation relation_with (const Constraint &c) const
  Returns the relations holding between the polyhedron `*this` *and the constraint* `c`.

- Poly_Gen_Relation relation_with (const Generator &g) const
  Returns the relations holding between the polyhedron `*this` *and the generator* `g`.

- Poly_Con_Relation relation_with (const Congruence &cg) const
  Returns the relations holding between the polyhedron `*this` *and the congruence* `c`.

- bool is_empty () const
  Returns `true` *if and only if* `*this` *is an empty polyhedron.*

- bool is_universe () const
  Returns `true` *if and only if* `*this` *is a universe polyhedron.*

- bool is_topologically_closed () const
  Returns `true` *if and only if* `*this` *is a topologically closed subset of the vector space.*

- bool is_disjoint_from (const Polyhedron &y) const
  Returns `true` *if and only if* `*this` *and* `y` *are disjoint.*

- bool is_discrete () const
  Returns `true` *if and only if* `*this` *is discrete.*

- bool is_bounded () const
  Returns `true` *if and only if* `*this` *is a bounded polyhedron.*

- bool contains_integer_point () const

Returns `true` *if and only if* `*this` *contains at least one integer point.*

- bool constrains (Variable var) const
  Returns `true` *if and only if* `var` *is constrained in* `*this`.

- bool bounds_from_above (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from above in* `*this`.

- bool bounds_from_below (const Linear_Expression &expr) const
  Returns `true` *if and only if* `expr` *is bounded from below in* `*this`.

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value is computed.*

- bool maximize (const Linear_Expression &expr, Coefficient &sup_n, Coefficient &sup_d, bool &maximum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from above in* `*this`*, in which case the supremum value and a point where* `expr` *reaches it are computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value is computed.*

- bool minimize (const Linear_Expression &expr, Coefficient &inf_n, Coefficient &inf_d, bool &minimum, Generator &g) const
  Returns `true` *if and only if* `*this` *is not empty and* `expr` *is bounded from below in* `*this`*, in which case the infimum value and a point where* `expr` *reaches it are computed.*

- bool frequency (const Linear_Expression &expr, Coefficient &freq_n, Coefficient &freq_d, Coefficient &val_n, Coefficient &val_d) const
  Returns `true` *if and only if there exist a unique value* `val` *such that* `*this` *saturates the equality* `expr = val`.

- bool contains (const Polyhedron &y) const
  Returns `true` *if and only if* `*this` *contains* `y`.

- bool strictly_contains (const Polyhedron &y) const
  Returns `true` *if and only if* `*this` *strictly contains* `y`.

- bool OK (bool check_not_empty=false) const
  Checks if all the invariants are satisfied.

Space Dimension Preserving Member Functions that May Modify the Polyhedron

- void add_constraint (const Constraint &c)
  Adds a copy of constraint `c` *to the system of constraints of* `*this` *(without minimizing the result).*

- void add_generator (const Generator &g)
  Adds a copy of generator `g` *to the system of generators of* `*this` *(without minimizing the result).*

- void add_congruence (const Congruence &cg)

Adds a copy of congruence `cg` to `*this`, if `cg` can be exactly represented by a polyhedron.

- void add_constraints (const Constraint_System &cs)
  Adds a copy of the constraints in `cs` to the system of constraints of `*this` (without minimizing the result).
- void add_recycled_constraints (Constraint_System &cs)
  Adds the constraints in `cs` to the system of constraints of `*this` (without minimizing the result).
- void add_generators (const Generator_System &gs)
  Adds a copy of the generators in `gs` to the system of generators of `*this` (without minimizing the result).
- void add_recycled_generators (Generator_System &gs)
  Adds the generators in `gs` to the system of generators of `*this` (without minimizing the result).
- void add_congruences (const Congruence_System &cgs)
  Adds a copy of the congruences in `cgs` to `*this`, if all the congruences can be exactly represented by a polyhedron.
- void add_recycled_congruences (Congruence_System &cgs)
  Adds the congruences in `cgs` to `*this`, if all the congruences can be exactly represented by a polyhedron.
- void refine_with_constraint (const Constraint &c)
  Uses a copy of constraint `c` to refine `*this`.
- void refine_with_congruence (const Congruence &cg)
  Uses a copy of congruence `cg` to refine `*this`.
- void refine_with_constraints (const Constraint_System &cs)
  Uses a copy of the constraints in `cs` to refine `*this`.
- void refine_with_congruences (const Congruence_System &cgs)
  Uses a copy of the congruences in `cgs` to refine `*this`.
- void unconstrain (Variable var)
  Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.
- void unconstrain (const Variables_Set &vars)
  Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.
- void intersection_assign (const Polyhedron &y)
  Assigns to `*this` the intersection of `*this` and `y`.
- void poly_hull_assign (const Polyhedron &y)
  Assigns to `*this` the poly-hull of `*this` and `y`.
- void upper_bound_assign (const Polyhedron &y)
  Same as poly_hull_assign(y).
- void poly_difference_assign (const Polyhedron &y)
  Assigns to `*this` the poly-difference of `*this` and `y`.
- void difference_assign (const Polyhedron &y)

Same as poly_difference_assign(y).

- bool simplify_using_context_assign (const Polyhedron &y)

  Assigns to `*this` *a meet-preserving simplification of* `*this` *with respect to* `y`*. If* `false` *is returned, then the intersection is empty.*

- void affine_image (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the affine image of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`*.*

- void affine_preimage (Variable var, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the affine preimage of* `*this` *under the function mapping variable* `var` *to the affine expression specified by* `expr` *and* `denominator`*.*

- void generalized_affine_image (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*

- void generalized_affine_preimage (Variable var, Relation_Symbol relsym, const Linear_Expression &expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*

- void generalized_affine_image (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the image of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*

- void generalized_affine_preimage (const Linear_Expression &lhs, Relation_Symbol relsym, const Linear_Expression &rhs)

  Assigns to `*this` *the preimage of* `*this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$*, where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*

- void bounded_affine_image (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the image of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$*.*

- void bounded_affine_preimage (Variable var, const Linear_Expression &lb_expr, const Linear_Expression &ub_expr, Coefficient_traits::const_reference denominator=Coefficient_one())

  Assigns to `*this` *the preimage of* `*this` *with respect to the bounded affine relation* $\frac{\mathrm{lb_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub_expr}}{\mathrm{denominator}}$*.*

- void time_elapse_assign (const Polyhedron &y)

  Assigns to `*this` *the result of computing the time-elapse between* `*this` *and* `y`*.*

- void wrap_assign (const Variables_Set &vars, Bounded_Integer_Type_Width w, Bounded_Integer_Type_Representation r, Bounded_Integer_Type_Overflow o, const Constraint_System *pcs=0, unsigned complexity_threshold=16, bool wrap_individually=true)

  Wraps the specified dimensions of the vector space.

- void drop_some_non_integer_points (Complexity_Class complexity=ANY_COMPLEXITY)

  Possibly tightens `*this` *by dropping some points with non-integer coordinates.*

- void drop_some_non_integer_points (const Variables_Set &vars, Complexity_Class complexity=ANY_COMPLEXITY)
  Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

- void topological_closure_assign ()
  Assigns to `*this` its topological closure.

- void BHRZ03_widening_assign (const Polyhedron &y, unsigned *tp=0)
  Assigns to `*this` the result of computing the BHRZ03-widening between `*this` and `y`.

- void limited_BHRZ03_extrapolation_assign (const Polyhedron &y, const Constraint_System &cs, unsigned *tp=0)
  Assigns to `*this` the result of computing the limited extrapolation between `*this` and `y` using the BHRZ03-widening operator.

- void bounded_BHRZ03_extrapolation_assign (const Polyhedron &y, const Constraint_System &cs, unsigned *tp=0)
  Assigns to `*this` the result of computing the bounded extrapolation between `*this` and `y` using the BHRZ03-widening operator.

- void H79_widening_assign (const Polyhedron &y, unsigned *tp=0)
  Assigns to `*this` the result of computing the H79_widening between `*this` and `y`.

- void widening_assign (const Polyhedron &y, unsigned *tp=0)
  Same as H79_widening_assign(y, tp).

- void limited_H79_extrapolation_assign (const Polyhedron &y, const Constraint_System &cs, unsigned *tp=0)
  Assigns to `*this` the result of computing the limited extrapolation between `*this` and `y` using the H79-widening operator.

- void bounded_H79_extrapolation_assign (const Polyhedron &y, const Constraint_System &cs, unsigned *tp=0)
  Assigns to `*this` the result of computing the bounded extrapolation between `*this` and `y` using the H79-widening operator.

Member Functions that May Modify the Dimension of the Vector Space

- void add_space_dimensions_and_embed (dimension_type m)
  Adds `m` new space dimensions and embeds the old polyhedron in the new vector space.

- void add_space_dimensions_and_project (dimension_type m)
  Adds `m` new space dimensions to the polyhedron and does not embed it in the new vector space.

- void concatenate_assign (const Polyhedron &y)
  Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

- void remove_space_dimensions (const Variables_Set &vars)
  Removes all the specified dimensions from the vector space.

- void remove_higher_space_dimensions (dimension_type new_dimension)
  Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`.

- template<typename Partial_Function >
  void map_space_dimensions (const Partial_Function &pfunc)
  Remaps the dimensions of the vector space according to a partial function.

- void expand_space_dimension (Variable var, dimension_type m)
  Creates `m` *copies of the space dimension corresponding to* `var`.

- void fold_space_dimensions (const Variables_Set &vars, Variable dest)
  Folds the space dimensions in `vars` *into* `dest`.

Miscellaneous Member Functions

- ~Polyhedron ()
  Destructor.

- void swap (Polyhedron &y)
  Swaps `*this` *with polyhedron* `y`. *(*`*this` *and* `y` *can be dimension-incompatible.).*

- void ascii_dump () const
  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const
  Writes to `s` *an ASCII representation of* `*this`.

- void print () const
  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)
  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const
  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.

- int32_t hash_code () const
  Returns a 32-bit hash code for `*this`.

Static Public Member Functions

- static dimension_type max_space_dimension ()
  Returns the maximum space dimension all kinds of Polyhedron can handle.

- static bool can_recycle_constraint_systems ()
  Returns `true` *indicating that this domain has methods that can recycle constraints.*

- static void initialize ()
  Initializes the class.

- static void finalize ()

  Finalizes the class.

- static bool can_recycle_congruence_systems ()

  Returns `false` indicating that this domain cannot recycle congruences.

Protected Member Functions

- Polyhedron (Topology topol, dimension_type num_dimensions, Degenerate_Element kind)

  Builds a polyhedron having the specified properties.

- Polyhedron (const Polyhedron &y, Complexity_Class complexity=ANY_COMPLEXITY)

  Ordinary copy constructor.

- Polyhedron (Topology topol, const Constraint_System &cs)

  Builds a polyhedron from a system of constraints.

- Polyhedron (Topology topol, Constraint_System &cs, Recycle_Input dummy)

  Builds a polyhedron recycling a system of constraints.

- Polyhedron (Topology topol, const Generator_System &gs)

  Builds a polyhedron from a system of generators.

- Polyhedron (Topology topol, Generator_System &gs, Recycle_Input dummy)

  Builds a polyhedron recycling a system of generators.

- template<typename Interval >
  Polyhedron (Topology topol, const Box< Interval > &box, Complexity_Class complexity=ANY_COMPLEXITY)

  Builds a polyhedron from a box.

- Polyhedron & operator= (const Polyhedron &y)

  The assignment operator. (`*this` and `y` can be dimension-incompatible.).

- bool BFT00_poly_hull_assign_if_exact (const Polyhedron &y)

  If the poly-hull of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

- bool BHZ09_poly_hull_assign_if_exact (const Polyhedron &y)
- bool BHZ09_C_poly_hull_assign_if_exact (const Polyhedron &y)
- bool BHZ09_NNC_poly_hull_assign_if_exact (const Polyhedron &y)
- void drop_some_non_integer_points (const Variables_Set *pvars, Complexity_Class complexity)

  Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `*pvars`.

Private Types

- enum Three_Valued_Boolean { TVB_TRUE, TVB_FALSE, TVB_DONT_KNOW }

Private Member Functions

- Topology topology () const
  Returns the topological kind of the polyhedron.
- bool is_necessarily_closed () const
  Returns `true` *if and only if the polyhedron is necessarily closed.*
- void refine_no_check (const Constraint &c)
  Uses a copy of constraint `c` *to refine the system of constraints of* `*this`.
- Three_Valued_Boolean quick_equivalence_test (const Polyhedron &y) const
  Polynomial but incomplete equivalence test between polyhedra.
- bool is_included_in (const Polyhedron &y) const
  Returns `true` *if and only if* `*this` *is included in* `y`.
- bool bounds (const Linear_Expression &expr, bool from_above) const
  Checks if and how `expr` *is bounded in* `*this`.
- bool max_min (const Linear_Expression &expr, bool maximize, Coefficient &ext_n, Coefficient &ext_d, bool &included, Generator &g) const
  Maximizes or minimizes `expr` *subject to* `*this`.

Private Verifiers: Verify if Individual Flags are Set

- bool marked_empty () const
  Returns `true` *if the polyhedron is known to be empty.*
- bool constraints_are_up_to_date () const
  Returns `true` *if the system of constraints is up-to-date.*
- bool generators_are_up_to_date () const
  Returns `true` *if the system of generators is up-to-date.*
- bool constraints_are_minimized () const
  Returns `true` *if the system of constraints is minimized.*
- bool generators_are_minimized () const
  Returns `true` *if the system of generators is minimized.*
- bool has_pending_constraints () const
  Returns `true` *if there are pending constraints.*
- bool has_pending_generators () const
  Returns `true` *if there are pending generators.*
- bool has_something_pending () const
  Returns `true` *if there are either pending constraints or pending generators.*
- bool can_have_something_pending () const
  Returns `true` *if the polyhedron can have something pending.*

- bool sat_c_is_up_to_date () const
  Returns `true` *if the saturation matrix* `sat_c` *is up-to-date.*

- bool sat_g_is_up_to_date () const
  Returns `true` *if the saturation matrix* `sat_g` *is up-to-date.*

State Flag Setters: Set Only the Specified Flags

- void set_zero_dim_univ ()
  Sets `status` *to express that the polyhedron is the universe 0-dimension vector space, clearing all corresponding matrices.*

- void set_empty ()
  Sets `status` *to express that the polyhedron is empty, clearing all corresponding matrices.*

- void set_constraints_up_to_date ()
  Sets `status` *to express that constraints are up-to-date.*

- void set_generators_up_to_date ()
  Sets `status` *to express that generators are up-to-date.*

- void set_constraints_minimized ()
  Sets `status` *to express that constraints are minimized.*

- void set_generators_minimized ()
  Sets `status` *to express that generators are minimized.*

- void set_constraints_pending ()
  Sets `status` *to express that constraints are pending.*

- void set_generators_pending ()
  Sets `status` *to express that generators are pending.*

- void set_sat_c_up_to_date ()
  Sets `status` *to express that* `sat_c` *is up-to-date.*

- void set_sat_g_up_to_date ()
  Sets `status` *to express that* `sat_g` *is up-to-date.*

State Flag Cleaners: Clear Only the Specified Flag

- void clear_empty ()
  Clears the `status` *flag indicating that the polyhedron is empty.*

- void clear_constraints_up_to_date ()
  Sets `status` *to express that constraints are no longer up-to-date.*

- void clear_generators_up_to_date ()
  Sets `status` *to express that generators are no longer up-to-date.*

- void clear_constraints_minimized ()
  Sets `status` *to express that constraints are no longer minimized.*

- void clear_generators_minimized ()
  Sets `status` to express that generators are no longer minimized.

- void clear_pending_constraints ()
  Sets `status` to express that there are no longer pending constraints.

- void clear_pending_generators ()
  Sets `status` to express that there are no longer pending generators.

- void clear_sat_c_up_to_date ()
  Sets `status` to express that `sat_c` is no longer up-to-date.

- void clear_sat_g_up_to_date ()
  Sets `status` to express that `sat_g` is no longer up-to-date.

The Handling of Pending Rows

- bool process_pending () const
  Processes the pending rows of either description of the polyhedron and obtains a minimized polyhedron.

- bool process_pending_constraints () const
  Processes the pending constraints and obtains a minimized polyhedron.

- void process_pending_generators () const
  Processes the pending generators and obtains a minimized polyhedron.

- void remove_pending_to_obtain_constraints () const
  Lazily integrates the pending descriptions of the polyhedron to obtain a constraint system without pending rows.

- bool remove_pending_to_obtain_generators () const
  Lazily integrates the pending descriptions of the polyhedron to obtain a generator system without pending rows.

Updating and Sorting Matrices

- void update_constraints () const
  Updates constraints starting from generators and minimizes them.

- bool update_generators () const
  Updates generators starting from constraints and minimizes them.

- void update_sat_c () const
  Updates `sat_c` using the updated constraints and generators.

- void update_sat_g () const
  Updates `sat_g` using the updated constraints and generators.

- void obtain_sorted_constraints () const
  Sorts the matrix of constraints keeping status consistency.

- void obtain_sorted_generators () const

Sorts the matrix of generators keeping status consistency.

- void obtain_sorted_constraints_with_sat_c () const
  Sorts the matrix of constraints and updates `sat_c`.

- void obtain_sorted_generators_with_sat_g () const
  Sorts the matrix of generators and updates `sat_g`.

Weak and Strong Minimization of Descriptions

- bool minimize () const
  Applies (weak) minimization to both the constraints and generators.

- bool strongly_minimize_constraints () const
  Applies strong minimization to the constraints of an NNC polyhedron.

- bool strongly_minimize_generators () const
  Applies strong minimization to the generators of an NNC polyhedron.

- Constraint_System simplified_constraints () const
  If constraints are up-to-date, obtain a simplified copy of them.

Widening- and Extrapolation-Related Functions

- void select_CH78_constraints (const Polyhedron &y, Constraint_System &cs_selected) const
  Copies to `cs_selection` the constraints of `y` corresponding to the definition of the CH78-widening of `*this` and `y`.

- void select_H79_constraints (const Polyhedron &y, Constraint_System &cs_selected, Constraint_System &cs_not_selected) const
  Splits the constraints of 'x' into two subsets, depending on whether or not they are selected to compute the H79-widening of `*this` and `y`.

- bool BHRZ03_combining_constraints (const Polyhedron &y, const BHRZ03_Certificate &y_cert, const Polyhedron &H79, const Constraint_System &x_minus_H79_con_sys)
- bool BHRZ03_evolving_points (const Polyhedron &y, const BHRZ03_Certificate &y_cert, const Polyhedron &H79)
- bool BHRZ03_evolving_rays (const Polyhedron &y, const BHRZ03_Certificate &y_cert, const Polyhedron &H79)

Static Private Member Functions

- static void add_space_dimensions (Linear_System &mat1, Linear_System &mat2, Bit_Matrix &sat1, Bit_Matrix &sat2, dimension_type add_dim)
  Adds new space dimensions to the given matrices.

Minimization-Related Static Member Functions

- static bool minimize (bool con_to_gen, Linear_System &source, Linear_System &dest, Bit_Matrix &sat)
  Builds and simplifies constraints from generators (or vice versa).

- static bool add_and_minimize (bool con_to_gen, Linear_System &source1, Linear_System &dest, Bit_Matrix &sat, const Linear_System &source2)

  Adds given constraints and builds minimized corresponding generators or vice versa.

- static bool add_and_minimize (bool con_to_gen, Linear_System &source, Linear_System &dest, Bit_Matrix &sat)

  Adds given constraints and builds minimized corresponding generators or vice versa. The given constraints are in `source`.

- static dimension_type conversion (Linear_System &source, dimension_type start, Linear_System &dest, Bit_Matrix &sat, dimension_type num_lines_or_equalities)

  Performs the conversion from constraints to generators and vice versa.

- static dimension_type simplify (Linear_System &mat, Bit_Matrix &sat)

  Uses Gauss' elimination method to simplify the result of `conversion()`.

Private Attributes

- Constraint_System con_sys

  The system of constraints.

- Generator_System gen_sys

  The system of generators.

- Bit_Matrix sat_c

  The saturation matrix having constraints on its columns.

- Bit_Matrix sat_g

  The saturation matrix having generators on its columns.

- Status status

  The status flags to keep track of the polyhedron's internal state.

- dimension_type space_dim

  The number of dimensions of the enclosing vector space.

Static Private Attributes

- static dimension_type * simplify_num_saturators_p = 0

  Pointer to an array used by simplify().

- static size_t simplify_num_saturators_size = 0

  Dimension of an array used by simplify().

Friends

- class Parma_Polyhedra_Library::Box
- class Parma_Polyhedra_Library::BD_Shape
- class Parma_Polyhedra_Library::Octagonal_Shape
- class Parma_Polyhedra_Library::Grid
- class Parma_Polyhedra_Library::BHRZ03_Certificate
- class Parma_Polyhedra_Library::H79_Certificate
- bool operator== (const Polyhedron &x, const Polyhedron &y)
- bool Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces (const Polyhedron &)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Polyhedron &ph)

  Output operator.

- bool operator!= (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *and* `y` *are different polyhedra.*

- void swap (Parma_Polyhedra_Library::Polyhedron &x, Parma_Polyhedra_Library::Polyhedron &y)

  Specializes `std::swap`.

- template<typename PH >
  bool poly_hull_assign_if_exact (PH &p, const PH &q)

  If the poly-hull of `p` *and* `q` *is exact it is assigned to* `p` *and* `true` *is returned, otherwise* `false` *is returned.*

Exception Throwers

- void throw_runtime_error (const char *method) const
- void throw_invalid_argument (const char *method, const char *reason) const
- void throw_topology_incompatible (const char *method, const char *ph_name, const Polyhedron &ph) const
- void throw_topology_incompatible (const char *method, const char *c_name, const Constraint &c) const
- void throw_topology_incompatible (const char *method, const char *g_name, const Generator &g) const
- void throw_topology_incompatible (const char *method, const char *cs_name, const Constraint_System &cs) const
- void throw_topology_incompatible (const char *method, const char *gs_name, const Generator_System &gs) const
- void throw_dimension_incompatible (const char *method, const char *other_name, dimension_type other_dim) const
- void throw_dimension_incompatible (const char *method, const char *ph_name, const Polyhedron &ph) const

- void throw_dimension_incompatible (const char *method, const char *e_name, const Linear_Expression &e) const
- void throw_dimension_incompatible (const char *method, const char *c_name, const Constraint &c) const
- void throw_dimension_incompatible (const char *method, const char *g_name, const Generator &g) const
- void throw_dimension_incompatible (const char *method, const char *cg_name, const Congruence &cg) const
- void throw_dimension_incompatible (const char *method, const char *cs_name, const Constraint_System &cs) const
- void throw_dimension_incompatible (const char *method, const char *gs_name, const Generator_System &gs) const
- void throw_dimension_incompatible (const char *method, const char *cgs_name, const Congruence_System &cgs) const
- void throw_dimension_incompatible (const char *method, const char *var_name, Variable var) const
- void throw_dimension_incompatible (const char *method, dimension_type required_space_dim) const
- void throw_invalid_generator (const char *method, const char *g_name) const
- void throw_invalid_generators (const char *method, const char *gs_name) const
- static void throw_space_dimension_overflow (Topology topol, const char *method, const char *reason)

11.160.1 Detailed Description

The base class for convex polyhedra. An object of the class Polyhedron represents a convex polyhedron in the vector space $\mathbb{R}^n$.

A polyhedron can be specified as either a finite system of constraints or a finite system of generators (see Section Representations of Convex Polyhedra) and it is always possible to obtain either representation. That is, if we know the system of constraints, we can obtain from this the system of generators that define the same polyhedron and vice versa. These systems can contain redundant members: in this case we say that they are not in the minimal form.

Two key attributes of any polyhedron are its topological kind (recording whether it is a C_Polyhedron or an NNC_Polyhedron object) and its space dimension (the dimension $n \in \mathbb{N}$ of the enclosing vector space):

- all polyhedra, the empty ones included, are endowed with a specific topology and space dimension;
- most operations working on a polyhedron and another object (i.e., another polyhedron, a constraint or generator, a set of variables, etc.) will throw an exception if the polyhedron and the object are not both topology-compatible and dimension-compatible (see Section Representations of Convex Polyhedra);
- the topology of a polyhedron cannot be changed; rather, there are constructors for each of the two derived classes that will build a new polyhedron with the topology of that class from another polyhedron from either class and any topology;
- the only ways in which the space dimension of a polyhedron can be changed are:
  - *explicit* calls to operators provided for that purpose;
  - standard copy, assignment and swap operators.

Note that four different polyhedra can be defined on the zero-dimension space: the empty polyhedron, either closed or NNC, and the universe polyhedron R^0, again either closed or NNC.

In all the examples it is assumed that variables x and y are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a polyhedron corresponding to a square in $\mathbb{R}^2$, given as a system of constraints:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
C_Polyhedron ph(cs);
```

The following code builds the same polyhedron as above, but starting from a system of generators specifying the four vertices of the square:

```
Generator_System gs;
gs.insert(point(0*x + 0*y));
gs.insert(point(0*x + 3*y));
gs.insert(point(3*x + 0*y));
gs.insert(point(3*x + 3*y));
C_Polyhedron ph(gs);
```

Example 2

The following code builds an unbounded polyhedron corresponding to a half-strip in $\mathbb{R}^2$, given as a system of constraints:

```
Constraint_System cs;
cs.insert(x >= 0);
cs.insert(x - y <= 0);
cs.insert(x - y + 1 >= 0);
C_Polyhedron ph(cs);
```

The following code builds the same polyhedron as above, but starting from the system of generators specifying the two vertices of the polyhedron and one ray:

```
Generator_System gs;
gs.insert(point(0*x + 0*y));
gs.insert(point(0*x + y));
gs.insert(ray(x - y));
C_Polyhedron ph(gs);
```

Example 3

The following code builds the polyhedron corresponding to a half-plane by adding a single constraint to the universe polyhedron in $\mathbb{R}^2$:

```
C_Polyhedron ph(2);
ph.add_constraint(y >= 0);
```

The following code builds the same polyhedron as above, but starting from the empty polyhedron in the space $\mathbb{R}^2$ and inserting the appropriate generators (a point, a ray and a line).

```
C_Polyhedron ph(2, EMPTY);
ph.add_generator(point(0*x + 0*y));
ph.add_generator(ray(y));
ph.add_generator(line(x));
```

Note that, although the above polyhedron has no vertices, we must add one point, because otherwise the result of the Minkowski's sum would be an empty polyhedron. To avoid subtle errors related to the minimization process, it is required that the first generator inserted in an empty polyhedron is a point (otherwise, an exception is thrown).

Example 4

The following code shows the use of the function `add_space_dimensions_and_embed`:

```
C_Polyhedron ph(1);
ph.add_constraint(x == 2);
ph.add_space_dimensions_and_embed(1);
```

We build the universe polyhedron in the 1-dimension space $\mathbb{R}$. Then we add a single equality constraint, thus obtaining the polyhedron corresponding to the singleton set $\{2\} \subseteq \mathbb{R}$. After the last line of code, the resulting polyhedron is

$$\left\{ (2, y)^{\mathrm{T}} \in \mathbb{R}^2 \;\middle|\; y \in \mathbb{R} \right\}.$$

Example 5

The following code shows the use of the function `add_space_dimensions_and_project`:

```
C_Polyhedron ph(1);
ph.add_constraint(x == 2);
ph.add_space_dimensions_and_project(1);
```

The first two lines of code are the same as in Example 4 for `add_space_dimensions_and_embed`. After the last line of code, the resulting polyhedron is the singleton set $\left\{(2, 0)^{\mathrm{T}}\right\} \subseteq \mathbb{R}^2$.

Example 6

The following code shows the use of the function `affine_image`:

```
C_Polyhedron ph(2, EMPTY);
ph.add_generator(point(0*x + 0*y));
ph.add_generator(point(0*x + 3*y));
ph.add_generator(point(3*x + 0*y));
ph.add_generator(point(3*x + 3*y));
Linear_Expression expr = x + 4;
ph.affine_image(x, expr);
```

In this example the starting polyhedron is a square in $\mathbb{R}^2$, the considered variable is x and the affine expression is $x + 4$. The resulting polyhedron is the same square translated to the right. Moreover, if the affine transformation for the same variable `x` is $x + y$:

```
Linear_Expression expr = x + y;
```

the resulting polyhedron is a parallelogram with the height equal to the side of the square and the oblique sides parallel to the line $x - y$. Instead, if we do not use an invertible transformation for the same variable; for example, the affine expression y:

```
Linear_Expression expr = y;
```

the resulting polyhedron is a diagonal of the square.

Example 7

The following code shows the use of the function `affine_preimage`:

```
C_Polyhedron ph(2);
ph.add_constraint(x >= 0);
ph.add_constraint(x <= 3);
ph.add_constraint(y >= 0);
ph.add_constraint(y <= 3);
Linear_Expression expr = x + 4;
ph.affine_preimage(x, expr);
```

In this example the starting polyhedron, `var` and the affine expression and the denominator are the same as in Example 6, while the resulting polyhedron is again the same square, but translated to the left. Moreover, if the affine transformation for `x` is $x + y$

```
Linear_Expression expr = x + y;
```

the resulting polyhedron is a parallelogram with the height equal to the side of the square and the oblique sides parallel to the line $x + y$. Instead, if we do not use an invertible transformation for the same variable x, for example, the affine expression y:

```
Linear_Expression expr = y;
```

the resulting polyhedron is a line that corresponds to the y axis.

Example 8

For this example we use also the variables:

```
Variable z(2);
Variable w(3);
```

The following code shows the use of the function `remove_space_dimensions`:

```
Generator_System gs;
gs.insert(point(3*x + y +0*z + 2*w));
C_Polyhedron ph(gs);
Variables_Set vars;
vars.insert(y);
vars.insert(z);
ph.remove_space_dimensions(vars);
```

The starting polyhedron is the singleton set $\{(3, 1, 0, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^4$, while the resulting polyhedron is $\{(3, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$. Be careful when removing space dimensions *incrementally*: since dimensions are automatically renamed after each application of the `remove_space_dimensions` operator, unexpected results can be obtained. For instance, by using the following code we would obtain a different result:

```
set<Variable> vars1;
vars1.insert(y);
ph.remove_space_dimensions(vars1);
set<Variable> vars2;
vars2.insert(z);
ph.remove_space_dimensions(vars2);
```

In this case, the result is the polyhedron $\{(3, 0)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$: when removing the set of dimensions `vars2` we are actually removing variable w of the original polyhedron. For the same reason, the operator `remove_space_dimensions` is not idempotent: removing twice the same non-empty set of dimensions is never the same as removing them just once.

Definition at line 365 of file Polyhedron.defs.hh.

11.160.2 Member Typedef Documentation

11.160.2.1 typedef Coefficient Parma_Polyhedra_Library::Polyhedron::coefficient_type

The numeric type of coefficients.

Definition at line 368 of file Polyhedron.defs.hh.

11.160.3 Member Enumeration Documentation

11.160.3.1 enum Parma_Polyhedra_Library::Polyhedron::Three_Valued_Boolean [private]

Enumerator:

TVB_TRUE

TVB_FALSE

TVB_DONT_KNOW

Definition at line 2296 of file Polyhedron.defs.hh.

11.160.4 Constructor & Destructor Documentation

11.160.4.1 Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, dimension_type *num_dimensions*, Degenerate_Element *kind*) [protected]

Builds a polyhedron having the specified properties.

Parameters

topol The topology of the polyhedron;

num_dimensions The number of dimensions of the vector space enclosing the polyhedron;

kind Specifies whether the universe or the empty polyhedron has to be built.

Definition at line 50 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), con_sys, Parma_Polyhedra_Library::EMPTY, max_space_dimension(), OK(), set_constraints_minimized(), Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), space_dim, and status.

11.160.4.2 Parma_Polyhedra_Library::Polyhedron::Polyhedron (const Polyhedron & *y*, Complexity_Class *complexity* = ANY_COMPLEXITY) [protected]

Ordinary copy constructor.

The complexity argument is ignored.

Definition at line 71 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::assign_with_pending(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), and topology().

11.160.4.3 Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, const Constraint_System & *cs*) [protected]

Builds a polyhedron from a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

topol The topology of the polyhedron;

cs The system of constraints defining the polyhedron.

Exceptions

std::invalid_argument Thrown if the topology of `cs` is incompatible with `topol`.

Definition at line 88 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), con_sys, max_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_System::num_pending_rows(), OK(), set_constraints_up_to_date(), set_empty(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), swap(), throw_topology_incompatible(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

11.160.4.4 Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, Constraint_System & *cs*, Recycle_Input *dummy*) [protected]

Builds a polyhedron recycling a system of constraints.

The polyhedron inherits the space dimension of the constraint system.

Parameters

topol The topology of the polyhedron;

cs The system of constraints defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the topology of `cs` is incompatible with `topol`.

Definition at line 137 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), con_sys, max_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), set_constraints_up_to_date(), set_empty(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), swap(), throw_topology_incompatible(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

11.160.4.5 Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, const Generator_System & *gs*) [protected]

Builds a polyhedron from a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

topol The topology of the polyhedron;

gs The system of generators defining the polyhedron.

Exceptions

std::invalid_argument Thrown if the topology of `gs` is incompatible with `topol`, or if the system of generators is not empty but has no points.

Definition at line 185 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), gen_sys, max_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_System::num_pending_rows(), OK(), Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), set_generators_up_to_date(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, Parma_Polyhedra_Library::Generator_System::space_dimension(), status, swap(), throw_invalid_generators(), throw_topology_incompatible(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

11.160.4.6 Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, Generator_System & *gs*, Recycle_Input *dummy*) [protected]

Builds a polyhedron recycling a system of generators.

The polyhedron inherits the space dimension of the generator system.

Parameters

topol The topology of the polyhedron;

gs The system of generators defining the polyhedron. It is not declared `const` because its data-structures may be recycled to build the polyhedron.

dummy A dummy tag to syntactically differentiate this one from the other constructors.

Exceptions

std::invalid_argument Thrown if the topology of `gs` is incompatible with `topol`, or if the system of generators is not empty but has no points.

Definition at line 248 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), gen_sys, Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Generator_System::has_points(), max_space_dimension(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED, Parma_Polyhedra_Library::Linear_System::num_pending_rows(), OK(), Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), set_generators_up_to_date(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, Parma_Polyhedra_Library::Generator_System::space_dimension(), status, swap(), throw_invalid_generators(), throw_topology_incompatible(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

11.160.4.7 template<typename Interval > Parma_Polyhedra_Library::Polyhedron::Polyhedron (Topology *topol*, const Box< Interval > & *box*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline, protected]

Builds a polyhedron from a box.

This will use an algorithm whose complexity is polynomial and build the smallest polyhedron with topology `topol` containing `box`.

Parameters

topol The topology of the polyhedron;

box The box representing the polyhedron to be built;

complexity This argument is ignored.

Definition at line 36 of file Polyhedron.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::add_low_level_constraints(), con_sys, Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Box< ITV >::get_lower_bound(), Parma_Polyhedra_Library::Box< ITV >::get_upper_bound(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Box< ITV >::is_empty(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, Parma_Polyhedra_Library::Matrix::num_rows(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, set_constraints_up_to_date(), set_empty(), Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row(), Parma_Polyhedra_Library::Linear_System::set_sorted(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Box< ITV >::space_dimension(), and Parma_Polyhedra_Library::Constraint_System::swap().

11.160.4.8 Parma_Polyhedra_Library::Polyhedron::~Polyhedron () [inline]

Destructor.

Definition at line 96 of file Polyhedron.inlines.hh.

11.160.5 Member Function Documentation

11.160.5.1 bool Parma_Polyhedra_Library::Polyhedron::add_and_minimize (bool *con_to_gen*, Linear_System & *source*, Linear_System & *dest*, Bit_Matrix & *sat*) [static, private]

Adds given constraints and builds minimized corresponding generators or vice versa. The given constraints are in `source`.

Returns

`true` if the obtained polyhedron is empty, `false` otherwise.

Parameters

con_to_gen `true` if `source` is a system of constraints, `false` otherwise;

source The first element of the given DD pair. It also contains the pending rows to be processed;

dest The second element of the given DD pair. It cannot have pending rows;

sat The saturation matrix that bind the upper part of `source` to `dest`.

On entry, the rows of `sat` are indexed by the rows of `dest` and its columns are indexed by the non-pending rows of `source`. On exit, the rows of `sat` are indexed by the rows of `dest` and its columns are indexed by the rows of `source`.

Let us suppose that `source` is a system of constraints. This method assumes that the non-pending part of `source` and system `dest` form a double description pair in minimal form and will build a new DD pair in minimal form by processing the pending constraints in `source`. To this end, it will call `conversion()`) and `simplify`.

This method treats also the dual case, i.e., processing pending generators. In this case `source` contains generators and `dest` is the system of constraints corresponding to the non-pending part of `source`.

Definition at line 335 of file minimize.cc.

References conversion(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_lines_or_equalities(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Bit_Matrix::resize(), and simplify().

11.160.5.2 bool Parma_Polyhedra_Library::Polyhedron::add_and_minimize (bool *con_to_gen*, Linear_System & *source1*, Linear_System & *dest*, Bit_Matrix & *sat*, const Linear_System & *source2*) `[static, private]`

Adds given constraints and builds minimized corresponding generators or vice versa.

Returns

`true` if the obtained polyhedron is empty, `false` otherwise.

Parameters

con_to_gen `true` if `source1` and `source2` are system of constraints, `false` otherwise;

source1 The first element of the given DD pair;

dest The second element of the given DD pair;

sat The saturation matrix that bind `source1` to `dest`;

source2 The new system of generators or constraints.

It is assumed that `source1` and `source2` are sorted and have no pending rows. It is also assumed that `dest` has no pending rows. On entry, the rows of `sat` are indexed by the rows of `dest` and its columns are indexed by the rows of `source1`. On exit, the rows of `sat` are indexed by the rows of `dest` and its columns are indexed by the rows of the system obtained by merging `source1` and `source2`.

Let us suppose we want to add some constraints to a given system of constraints `source1`. This method, given a minimized double description pair (`source1`, `dest`) and a system of new constraints `source2`, modifies `source1` by adding to it the constraints of `source2` that are not in `source1`. Then, by invoking `add_and_minimize(bool, Linear_System&, Linear_System&, Bit_Matrix&)`, processes the added constraints obtaining a new DD pair.

This method treats also the dual case, i.e., adding new generators to a previous system of generators. In this case `source1` contains the old generators, `source2` the new ones and `dest` is the system of constraints in the given minimized DD pair.

Since `source2` contains the constraints (or the generators) that will be added to `source1`, it is constant: it will not be modified.

Definition at line 234 of file minimize.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::cmp(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), and Parma_Polyhedra_Library::Matrix::num_rows().

Referenced by process_pending_constraints(), and process_pending_generators().

11.160.5.3 void Parma_Polyhedra_Library::Polyhedron::add_congruence (const Congruence & *cg*)

Adds a copy of congruence `cg` to `*this`, if `cg` can be exactly represented by a polyhedron.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible, of if `cg` is a proper congruence which is neither a tautology, nor a contradiction.

Definition at line 1274 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::Checked::le, marked_empty(), Parma_Polyhedra_Library::NECESSARILY_CLOSED, refine_no_check(), set_empty(), space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), throw_dimension_incompatible(), and throw_invalid_argument().

11.160.5.4 void Parma_Polyhedra_Library::Polyhedron::add_congruences (const Congruence_System & *cgs*)

Adds a copy of the congruences in `cgs` to `*this`, if all the congruences can be exactly represented by a polyhedron.

Parameters

cgs The congruences to be added.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, of if there exists in `cgs` a proper congruence which is neither a tautology, nor a contradiction.

Definition at line 1647 of file Polyhedron_public.cc.

References add_recycled_constraints(), Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), Parma_Polyhedra_Library::Congruence::is_proper_congruence(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::NECESSARILY_CLOSED, set_empty(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by add_recycled_congruences(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), and Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron().

11.160.5.5 void Parma_Polyhedra_Library::Polyhedron::add_constraint (const Constraint & *c*)

Adds a copy of constraint `c` to the system of constraints of `*this` (without minimizing the result).

Parameters

c The constraint that will be added to the system of constraints of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are topology-incompatible or dimension-incompatible.

Definition at line 1250 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint::is_inconsistent(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), Parma_Polyhedra_Library::Constraint::is_tautological(), marked_empty(), refine_no_check(), set_empty(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), throw_dimension_incompatible(), and throw_topology_incompatible().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition_aux().

11.160.5.6 void Parma_Polyhedra_Library::Polyhedron::add_constraints (const Constraint_System & *cs*)

Adds a copy of the constraints in `cs` to the system of constraints of `*this` (without minimizing the result).

Parameters

cs Contains the constraints that will be added to the system of constraints of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 1540 of file Polyhedron_public.cc.

References add_recycled_constraints().

Referenced by Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron(), and simplify_using_context_assign().

11.160.5.7 void Parma_Polyhedra_Library::Polyhedron::add_generator (const Generator & *g*)

Adds a copy of generator `g` to the system of generators of `*this` (without minimizing the result).

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are topology-incompatible or dimension-incompatible, or if `*this` is an empty polyhedron and `g` is not a point.

Definition at line 1312 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), can_have_something_pending(), clear_constraints_up_to_date(), clear_empty(), clear_generators_minimized(), Parma_Polyhedra_Library::Generator::divisor(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::Generator::line(), Parma_Polyhedra_Library::Generator::LINE, marked_empty(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Generator::point(), Parma_Polyhedra_Library::Generator::POINT, process_pending_constraints(), Parma_Polyhedra_Library::Generator::ray(), Parma_Polyhedra_Library::Generator::RAY, set_generators_minimized(), set_generators_pending(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Generator::space_dimension(), throw_dimension_incompatible(), throw_invalid_generator(), throw_runtime_error(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::Generator::type(), and update_generators().

Referenced by BFT00_poly_hull_assign_if_exact(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), and generalized_affine_image().

11.160.5.8 void Parma_Polyhedra_Library::Polyhedron::add_generators (const Generator_System & *gs*)

Adds a copy of the generators in `gs` to the system of generators of `*this` (without minimizing the result).

Parameters

gs Contains the generators that will be added to the system of generators of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `gs` are topology-incompatible or dimension-incompatible, or if `*this` is empty and the system of generators `gs` is not empty, but has no points.

Definition at line 1640 of file Polyhedron_public.cc.

References add_recycled_generators().

11.160.5.9 void Parma_Polyhedra_Library::Polyhedron::add_recycled_congruences (Congruence_System & *cgs*) [inline]

Adds the congruences in `cgs` to `*this`, if all the congruences can be exactly represented by a polyhedron.

Parameters

cgs The congruences to be added. Its elements may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible, of if there exists in `cgs` a proper congruence which is neither a tautology, nor a contradiction

Warning

The only assumption that can be made on `cgs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 372 of file Polyhedron.inlines.hh.

References add_congruences().

11.160.5.10 void Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints (Constraint_System & *cs*)

Adds the constraints in `cs` to the system of constraints of `*this` (without minimizing the result).

Parameters

cs The constraint system to be added to `*this`. The constraints in `cs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are topology-incompatible or dimension-incompatible.

Warning

The only assumption that can be made on `cs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 1447 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::begin(), can_have_something_pending(), clear_constraints_minimized(), clear_generators_up_to_date(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint_System::end(),

Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_generators(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Constraint::is_equality(), is_necessarily_closed(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_generators(), Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, set_constraints_pending(), Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), set_empty(), Parma_Polyhedra_Library::Constraint::set_is_equality(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, Parma_Polyhedra_Library::swap(), throw_dimension_incompatible(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), and update_constraints().

Referenced by add_congruences(), add_constraints(), BHRZ03_combining_constraints(), bounded_BHRZ03_extrapolation_assign(), bounded_H79_extrapolation_assign(), expand_space_dimension(), H79_widening_assign(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), refine_with_congruences(), and simplify_using_context_assign().

11.160.5.11 void Parma_Polyhedra_Library::Polyhedron::add_recycled_generators (Generator_System & *gs*)

Adds the generators in `gs` to the system of generators of `*this` (without minimizing the result).

Parameters

gs The generator system to be added to `*this`. The generators in `gs` may be recycled.

Exceptions

std::invalid_argument Thrown if `*this` and `gs` are topology-incompatible or dimension-incompatible, or if `*this` is empty and the system of generators `gs` is not empty, but has no points.

Warning

The only assumption that can be made on `gs` upon successful or exceptional return is that it can be safely destroyed.

Definition at line 1547 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Matrix::add_zero_rows(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), can_have_something_pending(), clear_constraints_up_to_date(), clear_empty(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Generator_System::has_closure_points(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_constraints(), Parma_Polyhedra_Library::Generator_System::has_points(), Parma_Polyhedra_Library::Generator::is_line(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_constraints(), Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, set_generators_pending(), set_generators_up_to_date(), Parma_Polyhedra_Library::Generator::set_is_line(), Parma_Polyhedra_Library::Linear_System::set_sorted(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Generator_System::space_dimension(), Parma_Polyhedra_Library::swap(), swap(), throw_dimension_incompatible(), throw_invalid_generators(), throw_topology_incompatible(), topology(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by add_generators(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), generalized_affine_image(), and generalized_affine_preimage().

11.160.5.12 void Parma_Polyhedra_Library::Polyhedron::add_space_dimensions (Linear_System & *mat1*, Linear_System & *mat2*, Bit_Matrix & *sat1*, Bit_Matrix & *sat2*, dimension_type *add_dim*) [static, private]

Adds new space dimensions to the given matrices.

Parameters

mat1 The matrix to which columns are added;

mat2 The matrix to which rows and columns are added;

sat1 The saturation matrix whose columns are indexed by the rows of matrix `mat1`. On entry it is up-to-date;

sat2 The saturation matrix whose columns are indexed by the rows of `mat2`;

add_dim The number of space dimensions to add.

Adds new space dimensions to the vector space modifying the matrices. This function is invoked only by `add_space_dimensions_and_embed()` and `add_space_dimensions_and_project()`, passing the matrix of constraints and that of generators (and the corresponding saturation matrices) in different order (see those methods for details).

Definition at line 36 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Bit_Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Bit_Matrix::num_rows(), Parma_Polyhedra_Library::Bit_Matrix::resize(), Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row(), swap(), Parma_Polyhedra_Library::swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Linear_System::topology(), and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

Referenced by add_space_dimensions_and_embed(), and add_space_dimensions_and_project().

11.160.5.13 void Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed (dimension_type *m*)

Adds `m` new space dimensions and embeds the old polyhedron in the new vector space.

Parameters

m The number of dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

The new space dimensions will be those having the highest indexes in the new polyhedron, which is characterized by a system of constraints in which the variables running through the new dimensions are not constrained. For instance, when starting from the polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the polyhedron

$$\left\{ (x, y, z)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{P} \right\}.$$

Definition at line 92 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), add_space_dimensions(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Constraint_System::clear(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), max_space_dimension(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), sat_c, sat_c_is_up_to_date(), sat_g, space_dim, space_dimension(), status, swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Polyhedron::Status::test_zero_dim_univ(), throw_space_dimension_overflow(), topology(), Parma_Polyhedra_Library::UNIVERSE, Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), and update_sat_c().

Referenced by bounded_affine_image(), bounded_affine_preimage(), expand_space_dimension(), generalized_affine_image(), and generalized_affine_preimage().

11.160.5.14 void Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project (dimension_type *m*)

Adds `m` new space dimensions to the polyhedron and does not embed it in the new vector space.

Parameters

m The number of space dimensions to add.

Exceptions

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

The new space dimensions will be those having the highest indexes in the new polyhedron, which is characterized by a system of constraints in which the variables running through the new dimensions are all constrained to be equal to 0. For instance, when starting from the polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ and adding a third space dimension, the result will be the polyhedron

$$\left\{ (x, y, 0)^{\mathrm{T}} \in \mathbb{R}^3 \;\middle|\; (x, y)^{\mathrm{T}} \in \mathcal{P} \right\}.$$

Definition at line 188 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), add_space_dimensions(), Parma_Polyhedra_Library::Matrix::add_zero_columns(), Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), Parma_Polyhedra_Library::Constraint_System::clear(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Generator_System::insert(), is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), max_space_dimension(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), sat_c, sat_g, sat_g_is_up_to_date(), set_generators_minimized(), space_dim, space_dimension(), status, swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), Parma_Polyhedra_Library::Polyhedron::Status::test_zero_dim_univ(), throw_space_dimension_overflow(), topology(),

Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), update_sat_g(), Parma_Polyhedra_-Library::Generator::zero_dim_closure_point(), and Parma_Polyhedra_Library::Generator::zero_dim_-point().

11.160.5.15 PPL::dimension_type Parma_Polyhedra_Library::Polyhedron::affine_dimension () const

Returns 0, if `*this` is empty; otherwise, returns the affine dimension of `*this`.

Definition at line 60 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_-Library::Constraint_System::end(), is_empty(), minimized_constraints(), and space_dim.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), BHZ09_C_-poly_hull_assign_if_exact(), and is_discrete().

11.160.5.16 void Parma_Polyhedra_Library::Polyhedron::affine_image (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the affine image of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is assigned;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

When considering the generators of a polyhedron, the affine transformation

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{\text{denominator}}$$

is assigned to `var` where `expr` is $\sum_{i=0}^{n-1} a_i x_i + b$ (b is the inhomogeneous term).

If constraints are up-to-date, it uses the specialized function affine_preimage() (for the system of constraints) and inverse transformation to reach the same result. To obtain the inverse transformation we use the following observation.

Observation:

1. The affine transformation is invertible if the coefficient of `var` in this transformation (i.e., a_{var}) is different from zero.

2. If the transformation is invertible, then we can write

$$\text{denominator} * {x'}_{\text{var}} = \sum_{i=0}^{n-1} a_i x_i + b = a_{\text{var}} x_{\text{var}} + \sum_{i \neq var} a_i x_i + b,$$

so that the inverse transformation is

$$a_{\text{var}} x_{\text{var}} = \text{denominator} * {x'}_{\text{var}} - \sum_{i \neq j} a_i x_i - b.$$

Then, if the transformation is invertible, all the entities that were up-to-date remain up-to-date. Otherwise only generators remain up-to-date.

In other words, if R is a $m_1 \times n$ matrix representing the rays of the polyhedron, V is a $m_2 \times n$ matrix representing the points of the polyhedron and

$$P = \left\{ \boldsymbol{x} = (x_0, \ldots, x_{n-1})^{\mathrm{T}} \;\middle|\; \boldsymbol{x} = \boldsymbol{\lambda} R + \boldsymbol{\mu} V, \boldsymbol{\lambda} \in \mathbb{R}_+^{m_1}, \boldsymbol{\mu} \in \mathbb{R}_+^{m_2}, \sum_{i=0}^{m_2-1} \mu_i = 1 \right\}$$

and T is the affine transformation to apply to P, then the resulting polyhedron is

$$P' = \left\{ (x_0, \ldots, T(x_0, \ldots, x_{n-1}), \ldots, x_{n-1})^{\mathrm{T}} \;\middle|\; (x_0, \ldots, x_{n-1})^{\mathrm{T}} \in P \right\}.$$

Affine transformations are, for example:

- translations
- rotations
- symmetries.

Definition at line 2557 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), clear_constraints_up_to_date(), clear_generators_minimized(), clear_sat_c_up_to_date(), clear_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), has_something_pending(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), OK(), remove_pending_to_obtain_generators(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by fold_space_dimensions(), and generalized_affine_image().

11.160.5.17 void Parma_Polyhedra_Library::Polyhedron::affine_preimage (Variable *var*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the affine preimage of `*this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var The variable to which the affine expression is substituted;

expr The numerator of the affine expression;

denominator The denominator of the affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

When considering constraints of a polyhedron, the affine transformation

$$\frac{\sum_{i=0}^{n-1} a_i x_i + b}{denominator},$$

is assigned to `var` where `expr` is $\sum_{i=0}^{n-1} a_i x_i + b$ (b is the inhomogeneous term).

If generators are up-to-date, then the specialized function affine_image() is used (for the system of generators) and inverse transformation to reach the same result. To obtain the inverse transformation, we use the following observation.

Observation:

1. The affine transformation is invertible if the coefficient of `var` in this transformation (i.e. a_{var}) is different from zero.
2. If the transformation is invertible, then we can write

$$\text{denominator} * {x'}_{\text{var}} = \sum_{i=0}^{n-1} a_i x_i + b = a_{\text{var}} x_{\text{var}} + \sum_{i \neq \text{var}} a_i x_i + b,$$

, the inverse transformation is

$$a_{\text{var}} x_{\text{var}} = \text{denominator} * {x'}_{\text{var}} - \sum_{i \neq j} a_i x_i - b.$$

.

Then, if the transformation is invertible, all the entities that were up-to-date remain up-to-date. Otherwise only constraints remain up-to-date.

In other words, if A is a $m \times n$ matrix representing the constraints of the polyhedron, T is the affine transformation to apply to P and

$$P = \left\{ \boldsymbol{x} = (x_0, \ldots, x_{n-1})^{\mathrm{T}} \;\middle|\; A\boldsymbol{x} \geq \boldsymbol{0} \right\}.$$

The resulting polyhedron is

$$P' = \left\{ \boldsymbol{x} = (x_0, \ldots, x_{n-1}))^{\mathrm{T}} \;\middle|\; A'\boldsymbol{x} \geq \boldsymbol{0} \right\},$$

where A' is defined as follows:

$${a'}_{ij} = \begin{cases} a_{ij} * \text{denominator} + a_{i\text{var}} * \text{expr}[j] & \text{for} j \neq \text{var}; \\ \text{expr}[\text{var}] * a_{i\text{var}}, & \text{for } j = \text{var}. \end{cases}$$

Definition at line 2637 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), clear_constraints_minimized(), clear_generators_up_to_date(), clear_sat_c_up_to_date(), clear_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(),

has_something_pending(), marked_empty(), minimize(), Parma_Polyhedra_Library::neg_assign(), OK(), remove_pending_to_obtain_constraints(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by generalized_affine_preimage().

11.160.5.18 void Parma_Polyhedra_Library::Polyhedron::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 3676 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Bit_Matrix::ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), Parma_Polyhedra_Library::Polyhedron::Status::ascii_dump(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), sat_c, sat_g, and status.

11.160.5.19 void Parma_Polyhedra_Library::Polyhedron::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by OK().

11.160.5.20 bool Parma_Polyhedra_Library::Polyhedron::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 3699 of file Polyhedron_public.cc.

11.160.5.21 bool Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact (const Polyhedron & *y*) [protected]

If the poly-hull of `*this` and `y` is exact it is assigned to `*this` and `true` is returned, otherwise `false` is returned.

Current implementation is based on (a variant of) Algorithm 8.1 in A. Bemporad, K. Fukuda, and F. D. Torrisi *Convexity Recognition of the Union of Polyhedra* Technical Report AUT00-13, ETH Zurich, 2000

Note

It is assumed that `*this` and `y` are topologically closed and dimension-compatible; if the assumption does not hold, the behavior is undefined.

Definition at line 1883 of file Polyhedron_nonpublic.cc.

References add_generator(), Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), con_sys, gen_sys, Parma_Polyhedra_Library::Poly_Con_Relation::implies(), is_empty(), Parma_-

Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Linear_Row::is_line_or_equality(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), Parma_Polyhedra_Library::Linear_Row::set_is_ray_or_point_or_inequality(), Parma_Polyhedra_Library::Linear_Row::sign_normalize(), Parma_Polyhedra_Library::Row::size(), space_dim, Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), topology(), and upper_bound_assign().

Referenced by Parma_Polyhedra_Library::C_Polyhedron::poly_hull_assign_if_exact().

11.160.5.22 bool Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints (const Polyhedron & *y*, const BHRZ03_Certificate & *y_cert*, const Polyhedron & *H79*, const Constraint_System & *x_minus_H79_con_sys*) [private]

Definition at line 385 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), Parma_Polyhedra_Library::Linear_Expression::all_homogeneous_terms_are_zero(), Parma_Polyhedra_Library::Constraint_System::clear(), con_sys, constraints_are_minimized(), contains(), gen_sys, generators_are_minimized(), has_something_pending(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Constraint::is_inequality(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::BHRZ03_Certificate::is_stabilizing(), marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), Parma_Polyhedra_Library::Scalar_Products::sign(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), swap(), Parma_Polyhedra_Library::Linear_System::topology(), and topology().

Referenced by BHRZ03_widening_assign().

11.160.5.23 bool Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_points (const Polyhedron & *y*, const BHRZ03_Certificate & *y_cert*, const Polyhedron & *H79*) [private]

Definition at line 507 of file Polyhedron_widenings.cc.

References add_recycled_generators(), constraints_are_minimized(), contains(), gen_sys, generators_are_minimized(), has_something_pending(), intersection_assign(), Parma_Polyhedra_Library::Generator::is_closure_point(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::BHRZ03_Certificate::is_stabilizing(), Parma_Polyhedra_Library::Linear_Row::linear_combine(), marked_empty(), minimize(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), space_dim, swap(), and topology().

Referenced by BHRZ03_widening_assign().

11.160.5.24 bool Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays (const Polyhedron & *y*, const BHRZ03_Certificate & *y_cert*, const Polyhedron & *H79*) [private]

Definition at line 577 of file Polyhedron_widenings.cc.

References add_recycled_generators(), constraints_are_minimized(), contains(), gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_something_pending(), Parma_Polyhedra_Library::Generator_System::insert(), intersection_assign(), Parma_Polyhedra_Library::Generator::is_ray(), Parma_Polyhedra_Library::BHRZ03_Certificate::is_stabilizing(), marked_empty(), minimize(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, relation_with(), space_dim, Parma_Polyhedra_Library::sub_mul_assign(), swap(), and topology().

Referenced by BHRZ03_widening_assign().

11.160.5.25 void Parma_Polyhedra_Library::Polyhedron::BHRZ03_widening_assign (const Polyhedron & *y*, unsigned * *tp* = 0)

Assigns to `*this` the result of computing the BHRZ03-widening between `*this` and `y`.

Parameters

y A polyhedron that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 666 of file Polyhedron_widenings.cc.

References BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), contains(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::BHRZ03_Certificate::is_stabilizing(), marked_empty(), minimize(), OK(), select_H79_constraints(), space_dim, swap(), throw_dimension_incompatible(), throw_topology_incompatible(), topology(), and Parma_Polyhedra_Library::UNIVERSE.

Referenced by limited_BHRZ03_extrapolation_assign().

11.160.5.26 bool Parma_Polyhedra_Library::Polyhedron::BHZ09_C_poly_hull_assign_if_exact (const Polyhedron & *y*) [protected]

Definition at line 1445 of file Polyhedron_nonpublic.cc.

References add_generator(), affine_dimension(), con_sys, gen_sys, Parma_Polyhedra_Library::Poly_Con_Relation::implies(), is_empty(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), is_included_in(), is_necessarily_closed(), minimize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), relation_with(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Bit_Row::set_until(), space_dim, Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), and update_sat_g().

Referenced by BHZ09_poly_hull_assign_if_exact().

11.160.5.27 bool Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact (const Polyhedron & *y*) [protected]

Definition at line 1548 of file Polyhedron_nonpublic.cc.

References add_constraint(), add_generator(), Parma_Polyhedra_Library::Bit_Row::clear(), con_sys, constraints(), contains(), Parma_Polyhedra_Library::Bit_Row::empty(), Parma_Polyhedra_Library::Bit_Row::first(), gen_sys, Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), is_empty(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::is_point(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), minimize(), Parma_Polyhedra_Library::Bit_Row::next(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), relation_with(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Bit_Row::set_until(), space_dim, Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), swap(), and update_sat_g().

Referenced by BHZ09_poly_hull_assign_if_exact().

11.160.5.28 bool Parma_Polyhedra_Library::Polyhedron::BHZ09_poly_hull_assign_if_exact (const Polyhedron & *y*) [protected]

Definition at line 1413 of file Polyhedron_nonpublic.cc.

References BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), is_empty(), is_necessarily_closed(), marked_empty(), space_dim, topology(), and upper_bound_assign().

Referenced by Parma_Polyhedra_Library::NNC_Polyhedron::poly_hull_assign_if_exact(), and Parma_Polyhedra_Library::C_Polyhedron::poly_hull_assign_if_exact().

11.160.5.29 void Parma_Polyhedra_Library::Polyhedron::bounded_affine_image (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = Coefficient_one())

Assigns to `*this` the image of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2713 of file Polyhedron_public.cc.

References add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::LESS_OR_EQUAL, marked_empty(), OK(), refine_no_check(), remove_higher_space_dimensions(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.160.5.30 void Parma_Polyhedra_Library::Polyhedron::bounded_affine_preimage (Variable *var*, const Linear_Expression & *lb_expr*, const Linear_Expression & *ub_expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the preimage of `*this` with respect to the bounded affine relation $\frac{\text{lb_expr}}{\text{denominator}} \leq \text{var}' \leq \frac{\text{ub_expr}}{\text{denominator}}$.

Parameters

var The variable updated by the affine relation;

lb_expr The numerator of the lower bounding affine expression;

ub_expr The numerator of the upper bounding affine expression;

denominator The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this`.

Definition at line 2788 of file Polyhedron_public.cc.

References add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), marked_empty(), OK(), Parma_Polyhedra_Library::Linear_System::permute_columns(), refine_no_check(), remove_higher_space_dimensions(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), throw_invalid_argument(), and unconstrain().

11.160.5.31 void Parma_Polyhedra_Library::Polyhedron::bounded_BHRZ03_extrapolation_assign (const Polyhedron & *y*, const Constraint_System & *cs*, unsigned * *tp* = `0`)

Assigns to `*this` the result of computing the bounded extrapolation between `*this` and `y` using the BHRZ03-widening operator.

Parameters

y A polyhedron that *must* be contained in `*this`;

cs The system of constraints used to improve the widened polyhedron;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 847 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), Parma_Polyhedra_Library::ANY_COMPLEXITY, and limited_-BHRZ03_extrapolation_assign().

11.160.5.32 void Parma_Polyhedra_Library::Polyhedron::bounded_H79_extrapolation_assign (const Polyhedron & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0)

Assigns to `*this` the result of computing the bounded extrapolation between `*this` and `y` using the H79-widening operator.

Parameters

y A polyhedron that *must* be contained in `*this`;

cs The system of constraints used to improve the widened polyhedron;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 372 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), Parma_Polyhedra_Library::ANY_COMPLEXITY, and limited_-H79_extrapolation_assign().

11.160.5.33 bool Parma_Polyhedra_Library::Polyhedron::bounds (const Linear_Expression & *expr*, bool *from_above*) const [private]

Checks if and how `expr` is bounded in `*this`.

Returns `true` if and only if `from_above` is `true` and `expr` is bounded from above in `*this`, or `from_above` is `false` and `expr` is bounded from below in `*this`.

Parameters

expr The linear expression to test;

from_above `true` if and only if the boundedness of interest is "from above".

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 519 of file Polyhedron_nonpublic.cc.

References gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), process_pending_constraints(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by bounds_from_above(), and bounds_from_below().

11.160.5.34 bool Parma_Polyhedra_Library::Polyhedron::bounds_from_above (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from above in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 306 of file Polyhedron.inlines.hh.

References bounds().

11.160.5.35 bool Parma_Polyhedra_Library::Polyhedron::bounds_from_below (const Linear_Expression & *expr*) const [inline]

Returns `true` if and only if `expr` is bounded from below in `*this`.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

Definition at line 311 of file Polyhedron.inlines.hh.

References bounds().

11.160.5.36 bool Parma_Polyhedra_Library::Polyhedron::can_have_something_pending () const [inline, private]

Returns `true` if the polyhedron can have something pending.

Definition at line 173 of file Polyhedron.inlines.hh.

References constraints_are_minimized(), generators_are_minimized(), sat_c_is_up_to_date(), and sat_g_is_up_to_date().

Referenced by add_generator(), add_recycled_constraints(), add_recycled_generators(), concatenate_assign(), intersection_assign(), poly_hull_assign(), refine_no_check(), refine_with_constraints(), time_elapse_assign(), topological_closure_assign(), and unconstrain().

11.160.5.37 bool Parma_Polyhedra_Library::Polyhedron::can_recycle_congruence_systems () [inline, static]

Returns `false` indicating that this domain cannot recycle congruences.

Definition at line 118 of file Polyhedron.inlines.hh.

11.160.5.38 bool Parma_Polyhedra_Library::Polyhedron::can_recycle_constraint_systems () [inline, static]

Returns `true` indicating that this domain has methods that can recycle constraints.

Definition at line 112 of file Polyhedron.inlines.hh.

11.160.5.39 void Parma_Polyhedra_Library::Polyhedron::clear_constraints_minimized () [inline, private]

Sets `status` to express that constraints are no longer minimized.

Definition at line 239 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_c_minimized(), and status.

Referenced by add_recycled_constraints(), affine_preimage(), clear_constraints_up_to_date(), concatenate_assign(), drop_some_non_integer_points(), intersection_assign(), refine_no_check(), refine_with_constraints(), remove_pending_to_obtain_constraints(), and topological_closure_assign().

11.160.5.40 void Parma_Polyhedra_Library::Polyhedron::clear_constraints_up_to_date () [inline, private]

Sets `status` to express that constraints are no longer up-to-date.

This also implies that they are neither minimized and both saturation matrices are no longer meaningful.

Definition at line 271 of file Polyhedron.inlines.hh.

References clear_constraints_minimized(), clear_pending_constraints(), clear_sat_c_up_to_date(), clear_-sat_g_up_to_date(), Parma_Polyhedra_Library::Polyhedron::Status::reset_c_up_to_date(), and status.

Referenced by add_generator(), add_recycled_generators(), affine_image(), generalized_affine_image(), poly_hull_assign(), remove_higher_space_dimensions(), remove_pending_to_obtain_generators(), remove_space_dimensions(), strongly_minimize_generators(), time_elapse_assign(), topological_-closure_assign(), and unconstrain().

11.160.5.41 void Parma_Polyhedra_Library::Polyhedron::clear_empty () [inline, private]

Clears the `status` flag indicating that the polyhedron is empty.

Definition at line 234 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_empty(), and status.

Referenced by add_generator(), and add_recycled_generators().

11.160.5.42 void Parma_Polyhedra_Library::Polyhedron::clear_generators_minimized () [inline, private]

Sets `status` to express that generators are no longer minimized.

Definition at line 244 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_g_minimized(), and status.

Referenced by add_generator(), add_recycled_generators(), affine_image(), clear_generators_up_to_-date(), generalized_affine_image(), poly_hull_assign(), remove_higher_space_dimensions(), remove_-pending_to_obtain_generators(), remove_space_dimensions(), time_elapse_assign(), topological_-closure_assign(), and unconstrain().

11.160.5.43 void Parma_Polyhedra_Library::Polyhedron::clear_generators_up_to_date () [inline, private]

Sets `status` to express that generators are no longer up-to-date.

This also implies that they are neither minimized and both saturation matrices are no longer meaningful.

Definition at line 281 of file Polyhedron.inlines.hh.

References clear_generators_minimized(), clear_pending_generators(), clear_sat_c_up_to_date(), clear_-sat_g_up_to_date(), Parma_Polyhedra_Library::Polyhedron::Status::reset_g_up_to_date(), and status.

Referenced by add_recycled_constraints(), affine_preimage(), concatenate_assign(), drop_some_non_-integer_points(), intersection_assign(), refine_no_check(), refine_with_constraints(), remove_pending_-to_obtain_constraints(), strongly_minimize_constraints(), and topological_closure_assign().

11.160.5.44 void Parma_Polyhedra_Library::Polyhedron::clear_pending_constraints () [inline, private]

Sets `status` to express that there are no longer pending constraints.

Definition at line 249 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_c_pending(), and status.

Referenced by clear_constraints_up_to_date(), process_pending_constraints(), and remove_pending_to_-obtain_constraints().

11.160.5.45 void Parma_Polyhedra_Library::Polyhedron::clear_pending_generators () [inline, private]

Sets `status` to express that there are no longer pending generators.

Definition at line 254 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_g_pending(), and status.

Referenced by clear_generators_up_to_date(), process_pending_generators(), and remove_pending_to_-obtain_generators().

11.160.5.46 void Parma_Polyhedra_Library::Polyhedron::clear_sat_c_up_to_date () [inline, private]

Sets `status` to express that `sat_c` is no longer up-to-date.

Definition at line 259 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_sat_c_up_to_date(), and status.

Referenced by affine_image(), affine_preimage(), clear_constraints_up_to_date(), clear_generators_-up_to_date(), concatenate_assign(), generalized_affine_image(), obtain_sorted_constraints(), process_-pending_generators(), and update_generators().

11.160.5.47 void Parma_Polyhedra_Library::Polyhedron::clear_sat_g_up_to_date () [inline, private]

Sets `status` to express that `sat_g` is no longer up-to-date.

Definition at line 265 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::reset_sat_g_up_to_date(), and status.

Referenced by affine_image(), affine_preimage(), clear_constraints_up_to_date(), clear_generators_-up_to_date(), concatenate_assign(), generalized_affine_image(), obtain_sorted_generators(), process_-pending_constraints(), and update_constraints().

11.160.5.48 void Parma_Polyhedra_Library::Polyhedron::concatenate_assign (const Polyhedron & *y*)

Assigns to `*this` the concatenation of `*this` and `y`, taken in this order.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible.

std::length_error Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

Definition at line 288 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Linear_System::add_rows_and_columns(), Parma_Polyhedra_-Library::Matrix::add_zero_rows_and_columns(), can_have_something_pending(), clear_constraints_-minimized(), clear_generators_up_to_date(), clear_sat_c_up_to_date(), clear_sat_g_up_to_date(),

con_sys, constraints(), constraints_are_up_to_date(), gen_sys, has_pending_generators(), Parma_Polyhedra_Library::Constraint::is_equality(), is_necessarily_closed(), marked_empty(), max_space_dimension(), Parma_Polyhedra_Library::Bit_Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Bit_Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_generators(), Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, Parma_Polyhedra_Library::Bit_Matrix::resize(), sat_c, sat_c_is_up_to_date(), sat_g, set_constraints_pending(), set_empty(), Parma_Polyhedra_Library::Constraint::set_is_equality(), set_sat_c_up_to_date(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_System::sort_rows(), space_dim, Parma_Polyhedra_Library::swap(), swap(), Parma_Polyhedra_Library::Matrix::swap_columns(), throw_space_dimension_overflow(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::Bit_Matrix::transpose_assign(), Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), and update_constraints().

11.160.5.49 Congruence_System Parma_Polyhedra_Library::Polyhedron::congruences () const [inline]

Returns a system of (equality) congruences satisfied by `*this`.

Definition at line 357 of file Polyhedron.inlines.hh.

References minimized_constraints().

11.160.5.50 bool Parma_Polyhedra_Library::Polyhedron::constrains (Variable *var*) const

Returns `true` if and only if `var` is constrained in `*this`.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 690 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Linear_Row::coefficient(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_rows(), process_pending_generators(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and update_constraints().

11.160.5.51 const PPL::Constraint_System & Parma_Polyhedra_Library::Polyhedron::constraints () const

Returns the system of constraints.

Definition at line 74 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint_System::adjust_topology_and_space_dimension(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(),

has_pending_constraints(), has_pending_generators(), marked_empty(), Parma_Polyhedra_-Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), obtain_sorted_-constraints(), process_pending_generators(), space_dim, Parma_Polyhedra_Library::Constraint_-System::space_dimension(), swap(), topology(), update_constraints(), and Parma_Polyhedra_-Library::Constraint_System::zero_dim_empty().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), BHZ09_NNC_poly_hull_-assign_if_exact(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::C_-Polyhedron::C_Polyhedron(), concatenate_assign(), contains_integer_point(), expand_space_dimension(), Parma_Polyhedra_Library::Grid::Grid(), minimized_constraints(), Parma_Polyhedra_Library::NNC_-Polyhedron::NNC_Polyhedron(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_-Shape(), and poly_difference_assign().

11.160.5.52 bool Parma_Polyhedra_Library::Polyhedron::constraints_are_minimized () const [inline, private]

Returns `true` if the system of constraints is minimized.

Note that only *weak* minimization is entailed, so that an NNC polyhedron may still have ϵ-redundant constraints.

Definition at line 138 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_c_minimized().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), BHRZ03_combining_-constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), can_have_something_pending(), Parma_Polyhedra_Library::Grid::Grid(), is_included_in(), is_universe(), minimize(), obtain_sorted_-constraints_with_sat_c(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), quick_equivalence_test(), select_CH78_constraints(), select_H79_constraints(), simplified_constraints(), simplify_using_context_assign(), update_sat_c(), and update_sat_g().

11.160.5.53 bool Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date () const [inline, private]

Returns `true` if the system of constraints is up-to-date.

Definition at line 128 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_c_up_to_date().

Referenced by add_recycled_constraints(), add_space_dimensions_and_embed(), add_space_-dimensions_and_project(), affine_image(), affine_preimage(), ascii_dump(), Parma_Polyhedra_-Library::BD_Shape< T >::BD_Shape(), bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::Box(), concatenate_assign(), constrains(), constraints(), drop_some_non_integer_points(), Parma_Polyhedra_Library::Grid::Grid(), H79_widening_assign(), intersection_assign(), is_included_in(), is_universe(), map_space_dimensions(), minimize(), obtain_sorted_constraints(), obtain_sorted_-constraints_with_sat_c(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), operator=(), Polyhedron(), refine_no_check(), refine_with_constraints(), relation_with(), select_H79_-constraints(), simplified_constraints(), simplify_using_context_assign(), topological_closure_assign(), and update_generators().

11.160.5.54 bool Parma_Polyhedra_Library::Polyhedron::contains (const Polyhedron & *y*) const

Returns `true` if and only if `*this` contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 3645 of file Polyhedron_public.cc.

References is_empty(), is_included_in(), marked_empty(), quick_equivalence_test(), space_dim, throw_dimension_incompatible(), throw_topology_incompatible(), topology(), and TVB_TRUE.

Referenced by BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), H79_widening_assign(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), poly_difference_assign(), poly_hull_assign_if_exact(), and strictly_contains().

11.160.5.55 bool Parma_Polyhedra_Library::Polyhedron::contains_integer_point () const

Returns `true` if and only if `*this` contains at least one integer point.

Definition at line 560 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::assign_r(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint::coefficient(), constraints(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::gcd_assign(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::is_canonical(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Checked::le, marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, process_pending(), space_dim, Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, and Parma_Polyhedra_Library::Constraint::type().

11.160.5.56 PPL::dimension_type Parma_Polyhedra_Library::Polyhedron::conversion (Linear_System & *source*, dimension_type *start*, Linear_System & *dest*, Bit_Matrix & *sat*, dimension_type *num_lines_or_equalities*) `[static, private]`

Performs the conversion from constraints to generators and vice versa.

Returns

The number of lines of the polyhedron or the number of equality constraints in the result of conversion.

Parameters

source The system to use to convert `dest`: it may be modified;

start The index of `source` row from which conversion begin;

dest The result of the conversion;

sat The saturation matrix telling us, for each row in `source`, which are the rows of `dest` that satisfy but do not saturate it;

num_lines_or_equalities The number of rows in the system `dest` that are either lines of the polyhedron (when `dest` is a system of generators) or equality constraints (when `dest` is a system of constraints).

For simplicity, all the following comments assume we are converting a constraint system `source` to a generator system `dest`; the comments for the symmetric case can be obtained by duality.

If some of the constraints in `source` are redundant, they will be removed. This is why the `source` is not declared to be a constant parameter.

If `start` is 0, then `source` is a sorted system; also, `dest` is a generator system corresponding to an empty constraint system. If otherwise `start` is greater than 0, then the two sub-systems of `source` made by the non-pending rows and the pending rows, respectively, are both sorted; also, `dest` is the generator system corresponding to the non-pending constraints of `source`.

Independently from the value of `start`, `dest` has lines from index 0 to index `num_lines_or_equalities` - 1 and rays/points from index `num_lines_or_equalities` to the last of its rows.

Note that here the rows of `sat` are indexed by rows of `dest` and its columns are indexed by rows of `source`.

We know that polyhedra can be represented by both a system of constraints or a system of generators (points, rays and lines) (see Section Representations of Convex Polyhedra). When we have both descriptions for a polyhedron P we have what is called a *double description* (or *DD pair*) for P.

Here, the *representation system* refers to the system C whose rows represent the constraints that characterize P and the *generating system*, the system G whose rows represent the generators of P. We say that a pair (C, G) of (real) systems is a *double description pair* if

$$C\boldsymbol{x} \geq \boldsymbol{0} \quad \Longleftrightarrow \quad \exists \boldsymbol{\lambda} \geq \boldsymbol{0} \,.\, \boldsymbol{x} = G\boldsymbol{\lambda}.$$

The term "double description" is quite natural in the sense that such a pair contains two different description of the same object. In fact, if we refer to the cone representation of a polyhedron P and we call C and G the systems of constraints and rays respectively, we have

$$P = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid C\boldsymbol{x} \geq \boldsymbol{0} \,\} = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} = G\boldsymbol{\lambda} \text{ for some } \boldsymbol{\lambda} \geq \boldsymbol{0} \,\}.$$

Because of the theorem of Minkowski (see Section Further Notation and Terminology), we can say that, given a $m \times n$ representation system C such that $\mathrm{rank}(C) = n = dimensionofthewholespace$ for a non-empty polyhedron P, it is always possible to find a generating system G for P such that (C, G) is a DD pair. Conversely, Weyl's theorem ensures that, for each generating system G, it is possible to find a representation system C such that (C, G) is a DD pair.

For efficiency reasons, our representation of polyhedra makes use of a double description. We are thus left with two problems:

1. given C find G such that (C, G) is a DD pair;
2. given G find C such that (C, G) is a DD pair.

Using Farkas' Lemma we can prove that these two problems are computationally equivalent (i.e., linear-time reducible to each other). Farkas' Lemma establishes a fundamental property of vectors in $\mathbb{R}^n$ that, in a sense, captures the essence of duality. Consider a matrix $A \in \mathbb{R}^{m \times n}$ and let $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ be its set of row vectors. Consider also another vector $\boldsymbol{c} \in \mathbb{R}^n$ such that, whenever a vector $\boldsymbol{y} \in \mathbb{R}^n$ has a non-negative

projection on the $\boldsymbol{a}_i$'s, it also has a non-negative projection on $\boldsymbol{c}$. The lemma states that $\boldsymbol{c}$ has this property if and only if it is in the cone generated by the $\boldsymbol{a}_i$'s. Formally, the lemma states the equivalence of the two following assertions:

1. $\forall \boldsymbol{y} : (A\boldsymbol{y} \geq 0 \implies \langle \boldsymbol{y}, \boldsymbol{c} \rangle \geq 0)$;
2. $\exists \boldsymbol{\lambda} \geq \boldsymbol{0} \,.\, \boldsymbol{c}^{\mathrm{T}} = \boldsymbol{\lambda}^{\mathrm{T}} A$.

With this result we can prove that (C, G) is a DD pair if and only if $(G^{\mathrm{T}}, C^{\mathrm{T}})$ is a DD pair.

Suppose (C, G) is a DD pair. Thus, for each x of the appropriate dimension, $C\boldsymbol{x} \geq \boldsymbol{0}$ if and only if $\exists \lambda \geq 0 \,.\, \boldsymbol{x} = G\boldsymbol{\lambda}$, which is of course equivalent to $\exists \boldsymbol{\lambda} \geq \boldsymbol{0} \,.\, \boldsymbol{x}^{\mathrm{T}} = \boldsymbol{\lambda}^{\mathrm{T}} G^{\mathrm{T}}$.

First, we assume that $\boldsymbol{z}$ is such that $G^{\mathrm{T}}\boldsymbol{z} \geq \boldsymbol{0}$ and we will show that $\exists \boldsymbol{\mu} \geq \boldsymbol{0} \,.\, \boldsymbol{z} = C^{\mathrm{T}}\boldsymbol{\mu}$. Let $\boldsymbol{x}$ be such that $C\boldsymbol{x} \geq \boldsymbol{0}$. Since (C, G) is a DD pair, this is equivalent to $\exists \boldsymbol{\lambda} \geq \boldsymbol{0} \,.\, \boldsymbol{x}^{\mathrm{T}} = \boldsymbol{\lambda}^{\mathrm{T}} G^{\mathrm{T}}$, which, by Farkas' Lemma is equivalent to $\forall \boldsymbol{y} : (G^{\mathrm{T}}\boldsymbol{y} \geq \boldsymbol{0} \implies \langle \boldsymbol{y}, \boldsymbol{x} \rangle \geq 0)$. Taking $\boldsymbol{y} = \boldsymbol{z}$ and recalling our assumption that $G^{\mathrm{T}}\boldsymbol{z} \geq \boldsymbol{0}$ we can conclude that $\langle \boldsymbol{z}, \boldsymbol{x} \rangle \geq 0$, that is equivalent to $\langle \boldsymbol{x}, \boldsymbol{z} \rangle \geq 0$. We have thus established that $\forall \boldsymbol{x} : (C\boldsymbol{x} \geq \boldsymbol{0} \implies \langle \boldsymbol{x}, \boldsymbol{z} \rangle \geq 0)$. By Farkas' Lemma, this is equivalent to $\exists \boldsymbol{\mu} \geq \boldsymbol{0} \,.\, \boldsymbol{z}^{\mathrm{T}} = \boldsymbol{\mu}^{\mathrm{T}} C$, which is equivalent to what we wanted to prove, that is, $\exists \boldsymbol{\mu} \geq \boldsymbol{0} \,.\, \boldsymbol{z} = C^{\mathrm{T}}\boldsymbol{\mu}$.

In order to prove the reverse implication, the following observation turns out to be useful: when (C, G) is a DD pair, $CG \geq 0$. In fact, let $\boldsymbol{e}_j$ be the vector whose components are all 0 apart from the j-th one, which is 1. Clearly $\boldsymbol{e}_j \geq \boldsymbol{0}$ and, taking $\boldsymbol{\lambda} = \boldsymbol{e}_j$ and $\boldsymbol{x} = G\boldsymbol{\lambda} = G\boldsymbol{e}_j$, we have $C\boldsymbol{x} = C(G\boldsymbol{e}_j) = (CG)\boldsymbol{e}_j \geq \boldsymbol{0}$, since (C, G) is a DD pair. Thus, as $(CG)\boldsymbol{e}_j$ is the j-th column of CG and since the choice of j was arbitrary, $CG \geq \boldsymbol{0}$.

We now assume that $\boldsymbol{z}$ is such that $\exists \boldsymbol{\mu} \geq \boldsymbol{0} \,.\, \boldsymbol{z} = C^{\mathrm{T}}\boldsymbol{\mu}$ and we will prove that $G^{\mathrm{T}}\boldsymbol{z} \geq \boldsymbol{0}$. By Farkas' Lemma, the assumption $\exists \boldsymbol{\mu} \geq \boldsymbol{0} \,.\, \boldsymbol{z}^{\mathrm{T}} = \boldsymbol{\mu}^{\mathrm{T}} C$, is equivalent to $\forall \boldsymbol{y} : (C\boldsymbol{y} \geq \boldsymbol{0} \implies \langle \boldsymbol{y}, \boldsymbol{z} \rangle \geq 0)$. If we take $\boldsymbol{y} = G\boldsymbol{e}_j$ then $C\boldsymbol{y} = CG\boldsymbol{e}_j \geq 0$, since $CG \geq \boldsymbol{0}$. So $\langle \boldsymbol{y}, \boldsymbol{z} \rangle = (\boldsymbol{e}_j^{\mathrm{T}} G^{\mathrm{T}})\boldsymbol{z} = \boldsymbol{e}_j^{\mathrm{T}}(G^{\mathrm{T}}\boldsymbol{z}) \geq 0$, that is, the j-th component of $G^{\mathrm{T}}\boldsymbol{z}$ is non-negative. The arbitrary choice of j allows us to conclude that $G^{\mathrm{T}}\boldsymbol{z} \geq \boldsymbol{0}$, as required.

In view of this result, the following exposition assumes, for clarity, that the conversion being performed is from constraints to generators. Thus, even if the roles of `source` and `dest` can be interchanged, in the sequel we assume the `source` system will contain the constraints that represent the polyhedron and the `dest` system will contain the generator that generates it.

There are some observations that are useful to understand this function:

Observation 1: Let A be a system of constraints that generate the polyhedron P and $\boldsymbol{c}$ a new constraint that must be added. Suppose that there is a line $\boldsymbol{z}$ that does not saturate the constraint $\boldsymbol{c}$. If we combine the old lines and rays that do not saturate $\boldsymbol{c}$ (except $\boldsymbol{z}$) with $\boldsymbol{z}$ such that the new ones saturate $\boldsymbol{c}$, the new lines and rays also saturate the constraints saturated by the old lines and rays.

In fact, if $\boldsymbol{y}_1$ is the old generator that does not saturate $\boldsymbol{c}$, $\boldsymbol{y}_2$ is the new one such that

$$\boldsymbol{y}_2 = \lambda \boldsymbol{y}_1 + \mu \boldsymbol{z}$$

and $\boldsymbol{c}_1$ is a previous constraint that $\boldsymbol{y}_1$ and $\boldsymbol{z}$ saturates, we can see

$$\langle \boldsymbol{c}_1, \boldsymbol{y}_2 \rangle = \langle \boldsymbol{c}_1, (\lambda \boldsymbol{y}_1 + \mu \boldsymbol{z}) \rangle = \lambda \langle \boldsymbol{c}_1, \boldsymbol{y}_1 \rangle + \mu \langle \boldsymbol{c}_1, \boldsymbol{z} \rangle = 0 + \mu \langle \boldsymbol{c}_1, \boldsymbol{z} \rangle = \mu \langle \boldsymbol{c}_1, \boldsymbol{z} \rangle$$

and

$$\mu \langle \boldsymbol{c}_1, \boldsymbol{z} \rangle = 0.$$

Proposition 1: Let $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ be distinct rays of P. Then the following statements are equivalent: a) $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are adjacent extreme rays (see Section Further Notation and Terminology); b) $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are extreme rays and the rank of the system composed by the constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is equal to $d - 2$, where d is the rank of the system of constraints.

In fact, let F be the system of generators that saturate the constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. If b) holds, the set F is 2-dimensional and $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ generate this set. So, every generator $\boldsymbol{x}$ of F can be built as a combination of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, i.e.

$$\boldsymbol{x} = \lambda \boldsymbol{r}_1 + \mu \boldsymbol{r}_2.$$

This combination is non-negative because there exists at least a constraint c saturated by $\boldsymbol{r}_1$ and not $\boldsymbol{r}_2$ (or vice versa) (because they are distinct) for which

$$\langle \boldsymbol{c}, \boldsymbol{x} \rangle \geq 0$$

and

$$\langle \boldsymbol{c}, \boldsymbol{x} \rangle = \lambda \langle \boldsymbol{c}, \boldsymbol{r}_1 \rangle (or = \mu \langle \boldsymbol{c}, \boldsymbol{r}_2 \rangle).$$

So, there is no other extreme ray in F and a) holds. Otherwise, if b) does not hold, the rank of the system generated by the constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is equal to $d - k$, with k >= 3, the set F is k -dimensional and at least k extreme rays are necessary to generate F. So, $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are not adjacent and a) does not hold.

Proposition 2: When we build the new system of generators starting from a system A of constraints of P, if $\boldsymbol{c}$ is the constraint to add to A and all lines of P saturate $\boldsymbol{c}$, the new set of rays is the union of those rays that saturate, of those that satisfy and of a set $\overline{Q}$ of rays such that each of them

1. lies on the hyper-plane represented by the k-th constraint,

2. is a positive combination of two adjacent rays $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ such that the first one satisfies the constraint and the other does not satisfy it. If the adjacency property is not taken in account, the new set of rays is not irredundant, in general.

In fact, if $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are not adjacent, the rank of the system composed by the constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is different from $d-2$ (see the previous proposition) or neither $\boldsymbol{r}_1$ nor $\boldsymbol{r}_2$ are extreme rays. Since the new ray $\boldsymbol{r}$ is a combination of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, it saturates the same constraints saturated by both $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. If the rank is less than $d-2$, the rank of the system composed by $\boldsymbol{c}$ (that is saturated by $\boldsymbol{r}$) and by the constraints of A saturated by $\boldsymbol{r}$ is less than $d-1$. It means that r is redundant (see Section Further Notation and Terminology). If neither $\boldsymbol{r}_1$ nor $\boldsymbol{r}_2$ are extreme rays, they belong to a 2-dimensional face containing exactly two extreme rays of P. These two adjacent rays build a ray equal to $\boldsymbol{r}$ and so $\boldsymbol{r}$ is redundant.

Definition at line 348 of file conversion.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Bit_Matrix::add_recycled_row(), Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Coefficient_zero(), Parma_Polyhedra_Library::Bit_Matrix::columns_erase_to_end(), Parma_Polyhedra_Library::Bit_Row::count_ones(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), Parma_Polyhedra_Library::Linear_Row::is_ray_or_point_or_inequality(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::maybe_abandon(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::normalize2(), Parma_Polyhedra_Library::Bit_Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Bit_Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, Parma_Polyhedra_Library::Linear_Row::RAY_OR_POINT_OR_INEQUALITY, Parma_Polyhedra_Library::Bit_Matrix::rows_erase_to_end(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), Parma_Polyhedra_Library::sub_mul_assign(), Parma_Polyhedra_Library::Bit_Matrix::swap(), swap(), Parma_Polyhedra_Library::Linear_System::topology(), Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

Referenced by add_and_minimize(), and minimize().

11.160.5.57 void Parma_Polyhedra_Library::Polyhedron::difference_assign (const Polyhedron & *y*) [inline]

Same as poly_difference_assign(y).

Definition at line 86 of file Polyhedron.inlines.hh.

References poly_difference_assign().

11.160.5.58 void Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points (const Variables_Set * *pvars*, Complexity_Class *complexity*) [protected]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `*pvars`.

Parameters

pvars When nonzero, points with non-integer coordinates for the variables/space-dimensions contained in `*pvars` can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 2078 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::ANY_COMPLEXITY, clear_constraints_minimized(), clear_generators_up_to_date(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint::epsilon_leq_one(), Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint::is_equality(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_tautological(), marked_empty(), Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), PPL_DIRTY_TEMP_COEFFICIENT, process_pending_constraints(), process_pending_generators(), set_empty(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, and update_constraints().

11.160.5.59 void Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points (const Variables_Set & *vars*, Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates for the space dimensions corresponding to `vars`.

Parameters

vars Points with non-integer coordinates for these variables/space-dimensions can be discarded.

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 395 of file Polyhedron.inlines.hh.

References drop_some_non_integer_points().

11.160.5.60 void Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points (Complexity_Class *complexity* = ANY_COMPLEXITY) [inline]

Possibly tightens `*this` by dropping some points with non-integer coordinates.

Parameters

complexity The maximal complexity of any algorithms used.

Note

Currently there is no optimality guarantee, not even if `complexity` is `ANY_COMPLEXITY`.

Definition at line 389 of file Polyhedron.inlines.hh.

Referenced by drop_some_non_integer_points().

11.160.5.61 void Parma_Polyhedra_Library::Polyhedron::expand_space_dimension (Variable *var*, dimension_type *m*)

Creates `m` copies of the space dimension corresponding to `var`.

Parameters

var The variable corresponding to the space dimension to be replicated;

m The number of replicas to be created.

Exceptions

std::invalid_argument Thrown if `var` does not correspond to a dimension of the vector space.

std::length_error Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

If `*this` has space dimension n, with $n > 0$, and `var` has space dimension $k \leq n$, then the k-th space dimension is expanded to `m` new space dimensions $n, n+1, \ldots, n+m-1$.

Definition at line 548 of file Polyhedron_chdims.cc.

References add_recycled_constraints(), add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_Library::Constraint::coefficient(), constraints(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Variable::id(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_nonstrict_inequality(), max_space_dimension(), OK(), space_dim, space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), throw_space_dimension_overflow(), and topology().

11.160.5.62 PPL::memory_size_type Parma_Polyhedra_Library::Polyhedron::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Definition at line 3747 of file Polyhedron_public.cc.

References con_sys, Parma_Polyhedra_Library::Bit_Matrix::external_memory_in_bytes(), Parma_Polyhedra_Library::Generator_System::external_memory_in_bytes(), Parma_Polyhedra_Library::Constraint_System::external_memory_in_bytes(), gen_sys, sat_c, and sat_g.

Referenced by total_memory_in_bytes().

11.160.5.63 void Parma_Polyhedra_Library::Polyhedron::finalize () [static]

Finalizes the class.

Definition at line 53 of file Polyhedron_public.cc.

References simplify_num_saturators_p, and simplify_num_saturators_size.

11.160.5.64 void Parma_Polyhedra_Library::Polyhedron::fold_space_dimensions (const Variables_Set & *vars*, Variable *dest*)

Folds the space dimensions in `vars` into `dest`.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be folded;

dest The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

If `*this` has space dimension n, with $n > 0$, `dest` has space dimension $k \leq n$, `vars` is a set of variables whose maximum space dimension is also less than or equal to n, and `dest` is not a member of `vars`, then the space dimensions corresponding to variables in `vars` are folded into the k-th space dimension.

Definition at line 604 of file Polyhedron_chdims.cc.

References affine_image(), generators(), Parma_Polyhedra_Library::Variable::id(), marked_empty(), OK(), poly_hull_assign(), remove_space_dimensions(), space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.160.5.65 bool Parma_Polyhedra_Library::Polyhedron::frequency (const Linear_Expression & *expr*, Coefficient & *freq_n*, Coefficient & *freq_d*, Coefficient & *val_n*, Coefficient & *val_d*) const

Returns `true` if and only if there exist a unique value `val` such that `*this` saturates the equality `expr = val`.

Parameters

expr The linear expression for which the frequency is needed;

freq_n If `true` is returned, the value is set to 0; Present for interface compatibility with class Grid, where the frequency can have a non-zero value;

freq_d If `true` is returned, the value is set to 1;

val_n The numerator of `val`;

val_d The denominator of `val`;

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `false` is returned, then `freq_n`, `freq_d`, `val_n` and `val_d` are left untouched.

Definition at line 3462 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::assign_r(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Generator::is_closure_point(), is_empty(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Generator::is_point(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, process_pending_constraints(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and update_generators().

11.160.5.66 void Parma_Polyhedra_Library::Polyhedron::generalized_affine_image (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*)

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 3054 of file Polyhedron_public.cc.

References add_recycled_generators(), add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Generator_System::insert(), is_empty(), is_necessarily_closed(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, marked_empty(), Parma_Polyhedra_Library::NOT_EQUAL, OK(), refine_no_check(), remove_higher_space_dimensions(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.160.5.67 void Parma_Polyhedra_Library::Polyhedron::generalized_affine_image (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the image of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 2861 of file Polyhedron_public.cc.

References add_generator(), Parma_Polyhedra_Library::Linear_System::add_row(), affine_image(), clear_constraints_up_to_date(), clear_generators_minimized(), clear_sat_c_up_to_date(), clear_sat_g_up_to_date(), Parma_Polyhedra_Library::EQUAL, gen_sys, Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, is_empty(), is_necessarily_closed(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, minimize(), Parma_Polyhedra_Library::NOT_EQUAL, Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

Referenced by bounded_affine_image(), and generalized_affine_preimage().

11.160.5.68 void Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage (const Linear_Expression & *lhs*, Relation_Symbol *relsym*, const Linear_Expression & *rhs*)

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs The left hand side affine expression;

relsym The relation symbol;

rhs The right hand side affine expression.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with `lhs` or `rhs` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 3208 of file Polyhedron_public.cc.

References add_recycled_generators(), add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, Parma_Polyhedra_Library::Generator_System::insert(), is_empty(), is_necessarily_closed(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, marked_empty(), Parma_Polyhedra_Library::NOT_EQUAL, OK(), refine_no_check(), remove_higher_space_dimensions(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and throw_invalid_argument().

11.160.5.69 void Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage (Variable *var*, Relation_Symbol *relsym*, const Linear_Expression & *expr*, Coefficient_traits::const_reference *denominator* = `Coefficient_one()`)

Assigns to `*this` the preimage of `*this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var The left hand side variable of the generalized affine relation;

relsym The relation symbol;

expr The numerator of the right hand side affine expression;

denominator The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

std::invalid_argument Thrown if `denominator` is zero or if `expr` and `*this` are dimension-incompatible or if `var` is not a space dimension of `*this` or if `*this` is a C_Polyhedron and `relsym` is a strict relation symbol.

Definition at line 2953 of file Polyhedron_public.cc.

References affine_preimage(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::EQUAL, generalized_affine_image(), Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN, is_necessarily_closed(), Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::neg_assign(), Parma_Polyhedra_Library::NOT_EQUAL, OK(), PPL_DIRTY_TEMP_COEFFICIENT, refine_no_check(), Parma_Polyhedra_Library::Variable::space_dimension(), Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), throw_invalid_argument(), and unconstrain().

11.160.5.70 const PPL::Generator_System & Parma_Polyhedra_Library::Polyhedron::generators () const

Returns the system of generators.

Definition at line 130 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::adjust_topology_and_space_dimension(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_constraints(), has_pending_generators(), is_necessarily_closed(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), obtain_sorted_generators(), process_pending_constraints(), space_dim, Parma_Polyhedra_Library::Generator_System::space_dimension(), swap(), topology(), update_generators(), and Parma_Polyhedra_Library::Generator_System::zero_dim_univ().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::Box< ITV >::Box(), fold_space_dimensions(), Parma_Polyhedra_Library::Grid::Grid(), map_space_dimensions(), minimized_generators(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.160.5.71 bool Parma_Polyhedra_Library::Polyhedron::generators_are_minimized () const [inline, private]

Returns `true` if the system of generators is minimized.

Note that only *weak* minimization is entailed, so that an NNC polyhedron may still have ϵ-redundant generators.

Definition at line 143 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_g_minimized().

Referenced by BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), can_have_something_pending(), constrains(), generators(), is_topologically_closed(), is_universe(), minimize(), OK(), quick_equivalence_test(), simplify_using_context_assign(), update_sat_c(), and update_sat_g().

11.160.5.72 bool Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date () const [inline, private]

Returns `true` if the system of generators is up-to-date.

Definition at line 133 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_g_up_to_date().

Referenced by add_generator(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), bounded_affine_preimage(), bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), constrains(), contains_integer_point(), frequency(), generators(), Parma_Polyhedra_Library::Grid::Grid(), is_bounded(), is_empty(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), max_min(), minimize(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), operator=(), poly_hull_assign(), Polyhedron(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), time_elapse_assign(), topological_closure_assign(), unconstrain(), and update_constraints().

11.160.5.73 PPL::Grid_Generator_System Parma_Polyhedra_Library::Polyhedron::grid_generators () const

Returns a universe system of grid generators.

Definition at line 198 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Grid_Generator_System::insert(), and space_dim.

Referenced by minimized_grid_generators().

11.160.5.74 void Parma_Polyhedra_Library::Polyhedron::H79_widening_assign (const Polyhedron & *y*, unsigned * *tp* = 0)

Assigns to `*this` the result of computing the H79_widening between `*this` and `y`.

Parameters

y A polyhedron that *must* be contained in `*this`;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 157 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), con_sys, constraints_are_up_to_date(), contains(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_generators(), intersection_assign(), is_empty(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_generators(), select_CH78_constraints(), select_H79_constraints(), space_dim, swap(), throw_dimension_incompatible(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::UNIVERSE, and update_constraints().

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::H79_widening_assign(), limited_H79_extrapolation_assign(), and widening_assign().

11.160.5.75 bool Parma_Polyhedra_Library::Polyhedron::has_pending_constraints () const [inline, private]

Returns `true` if there are pending constraints.

Definition at line 158 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_c_pending().

Referenced by add_generator(), add_recycled_generators(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), constrains(), constraints(), contains_integer_point(), drop_some_non_integer_points(), frequency(), generators(),

intersection_assign(), is_bounded(), is_empty(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), poly_hull_assign(), process_pending(), process_pending_constraints(), process_pending_generators(), relation_with(), remove_pending_to_obtain_constraints(), remove_pending_to_obtain_generators(), select_CH78_constraints(), simplified_constraints(), time_elapse_assign(), topological_closure_assign(), and unconstrain().

11.160.5.76 bool Parma_Polyhedra_Library::Polyhedron::has_pending_generators () const [inline, private]

Returns `true` if there are pending generators.

Definition at line 163 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_g_pending().

Referenced by add_recycled_constraints(), concatenate_assign(), constrains(), constraints(), drop_some_non_integer_points(), generators(), H79_widening_assign(), intersection_assign(), is_included_in(), is_universe(), OK(), poly_hull_assign(), process_pending(), process_pending_constraints(), process_pending_generators(), refine_no_check(), refine_with_constraints(), relation_with(), remove_pending_to_obtain_constraints(), remove_pending_to_obtain_generators(), select_H79_constraints(), simplify_using_context_assign(), and topological_closure_assign().

11.160.5.77 bool Parma_Polyhedra_Library::Polyhedron::has_something_pending () const [inline, private]

Returns `true` if there are either pending constraints or pending generators.

Definition at line 168 of file Polyhedron.inlines.hh.

References status, Parma_Polyhedra_Library::Polyhedron::Status::test_c_pending(), and Parma_Polyhedra_Library::Polyhedron::Status::test_g_pending().

Referenced by affine_image(), affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), is_topologically_closed(), minimize(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), process_pending(), quick_equivalence_test(), remove_higher_space_dimensions(), remove_pending_to_obtain_constraints(), remove_pending_to_obtain_generators(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), simplify_using_context_assign(), update_constraints(), and update_generators().

11.160.5.78 int32_t Parma_Polyhedra_Library::Polyhedron::hash_code () const [inline]

Returns a 32-bit hash code for `*this`.

If `x` and `y` are such that `x == y`, then `x.hash_code() == y.hash_code()`.

Definition at line 45 of file Polyhedron.inlines.hh.

References space_dimension().

11.160.5.79 void Parma_Polyhedra_Library::Polyhedron::initialize () [static]

Initializes the class.

Definition at line 46 of file Polyhedron_public.cc.

References simplify_num_saturators_p, and simplify_num_saturators_size.

11.160.5.80 void Parma_Polyhedra_Library::Polyhedron::intersection_assign (const Polyhedron & *y*)

Assigns to `*this` the intersection of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 1900 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_rows(), Parma_Polyhedra_Library::Linear_System::add_rows(), can_have_something_pending(), clear_constraints_minimized(), clear_generators_up_to_date(), con_sys, constraints_are_up_to_date(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), OK(), process_pending_generators(), set_constraints_pending(), set_empty(), space_dim, throw_dimension_incompatible(), throw_topology_incompatible(), topology(), and update_constraints().

Referenced by BHRZ03_evolving_points(), BHRZ03_evolving_rays(), H79_widening_assign(), and is_disjoint_from().

11.160.5.81 bool Parma_Polyhedra_Library::Polyhedron::is_bounded () const

Returns `true` if and only if `*this` is a bounded polyhedron.

Definition at line 494 of file Polyhedron_public.cc.

References gen_sys, generators_are_up_to_date(), has_pending_constraints(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), process_pending_constraints(), space_dim, and update_generators().

11.160.5.82 bool Parma_Polyhedra_Library::Polyhedron::is_discrete () const [inline]

Returns `true` if and only if `*this` is discrete.

Definition at line 69 of file Polyhedron.inlines.hh.

References affine_dimension().

11.160.5.83 bool Parma_Polyhedra_Library::Polyhedron::is_disjoint_from (const Polyhedron & *y*) const

Returns `true` if and only if `*this` and `y` are disjoint.

Exceptions

std::invalid_argument Thrown if `x` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 3669 of file Polyhedron_public.cc.

References intersection_assign(), and is_empty().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment().

11.160.5.84 bool Parma_Polyhedra_Library::Polyhedron::is_empty () const [inline]

Returns `true` if and only if `*this` is an empty polyhedron.

Definition at line 180 of file Polyhedron.inlines.hh.

References generators_are_up_to_date(), has_pending_constraints(), marked_empty(), and minimize().

Referenced by affine_dimension(), BFT00_poly_hull_assign_if_exact(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), BHZ09_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), contains(), frequency(), generalized_affine_image(), generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::Grid(), H79_widening_assign(), is_disjoint_from(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::linear_partition_aux(), operator<<(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), and simplify_using_context_assign().

11.160.5.85 bool Parma_Polyhedra_Library::Polyhedron::is_included_in (const Polyhedron & *y*) const [private]

Returns `true` if and only if `*this` is included in `y`.

Definition at line 395 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator::CLOSURE_POINT, con_sys, constraints_are_minimized(), constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint::EQUALITY, has_pending_generators(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::LINE, marked_empty(), minimize(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Generator::POINT, process_pending_generators(), Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), Parma_Polyhedra_Library::Scalar_Products::sign(), space_dim, Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, topology(), Parma_Polyhedra_Library::Generator::type(), Parma_Polyhedra_Library::Constraint::type(), and update_constraints().

Referenced by BHZ09_C_poly_hull_assign_if_exact(), and contains().

11.160.5.86 bool Parma_Polyhedra_Library::Polyhedron::is_necessarily_closed () const [inline, private]

Returns `true` if and only if the polyhedron is necessarily closed.

Definition at line 74 of file Polyhedron.inlines.hh.

References con_sys, and Parma_Polyhedra_Library::Linear_System::is_necessarily_closed().

Referenced by add_constraint(), add_generator(), add_recycled_constraints(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), BHZ09_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), concatenate_assign(), drop_some_non_integer_points(), generalized_affine_image(), generalized_affine_preimage(), generators(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), H79_widening_assign(), Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces(), is_topologically_closed(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), minimized_constraints(), minimized_generators(), OK(), poly_difference_assign(), refine_no_check(), refine_with_constraints(), remove_higher_space_dimensions(), select_H79_constraints(), simplify_using_context_assign(), strongly_minimize_constraints(), strongly_minimize_generators(), throw_dimension_incompatible(), throw_invalid_argument(), throw_invalid_generator(), throw_invalid_generators(), throw_runtime_error(), throw_topology_incompatible(), time_elapse_assign(), topological_closure_assign(), and wrap_assign().

11.160.5.87 bool Parma_Polyhedra_Library::Polyhedron::is_topologically_closed () const

Returns `true` if and only if `*this` is a topologically closed subset of the vector space.

Definition at line 514 of file Polyhedron_public.cc.

References con_sys, gen_sys, generators_are_minimized(), has_something_pending(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Generator::is_matching_closure_point(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::is_point(), marked_empty(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Matrix::num_rows(), process_pending(), space_dim, and strongly_minimize_constraints().

11.160.5.88 bool Parma_Polyhedra_Library::Polyhedron::is_universe () const

Returns `true` if and only if `*this` is a universe polyhedron.

Definition at line 370 of file Polyhedron_public.cc.

References con_sys, constraints_are_minimized(), constraints_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), has_pending_constraints(), has_pending_generators(), is_necessarily_closed(), Parma_Polyhedra_Library::Generator::LINE, marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), obtain_sorted_constraints(), process_pending_generators(), Parma_Polyhedra_Library::Generator::RAY, and space_dim.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), and Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape().

11.160.5.89 void Parma_Polyhedra_Library::Polyhedron::limited_BHRZ03_extrapolation_assign (const Polyhedron & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0)

Assigns to `*this` the result of computing the limited extrapolation between `*this` and `y` using the BHRZ03-widening operator.

Parameters

y A polyhedron that *must* be contained in `*this`;

cs The system of constraints used to improve the widened polyhedron;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 763 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), BHRZ03_widening_assign(), contains(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Constraint_System::has_strict_inequalities(), Parma_Polyhedra_Library::Constraint_System::insert(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_constraints(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), throw_dimension_incompatible(), throw_topology_incompatible(), and update_generators().

Referenced by bounded_BHRZ03_extrapolation_assign().

11.160.5.90 void Parma_Polyhedra_Library::Polyhedron::limited_H79_extrapolation_assign (const Polyhedron & *y*, const Constraint_System & *cs*, unsigned * *tp* = 0)

Assigns to `*this` the result of computing the limited extrapolation between `*this` and `y` using the H79-widening operator.

Parameters

y A polyhedron that *must* be contained in `*this`;

cs The system of constraints used to improve the widened polyhedron;

tp An optional pointer to an unsigned variable storing the number of available tokens (to be used when applying the widening with tokens delay technique).

Exceptions

std::invalid_argument Thrown if `*this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

Definition at line 288 of file Polyhedron_widenings.cc.

References add_recycled_constraints(), contains(), gen_sys, generators_are_up_to_date(), H79_-widening_assign(), has_pending_constraints(), Parma_Polyhedra_Library::Constraint_System::has_-strict_inequalities(), Parma_Polyhedra_Library::Constraint_System::insert(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_-constraints(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), throw_dimension_incompatible(), throw_topology_incompatible(), and update_generators().

Referenced by bounded_H79_extrapolation_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::limited_H79_extrapolation_assign().

11.160.5.91 template<typename Partial_Function > void Parma_Polyhedra_-Library::Polyhedron::map_space_dimensions (const Partial_Function & *pfunc*) `[inline]`

Remaps the dimensions of the vector space according to a partial function.

Parameters

pfunc The partial function specifying the destiny of each space dimension.

The template type parameter Partial_Function must provide the following methods.

```
bool has_empty_codomain() const
```

returns `true` if and only if the represented partial function has an empty codomain (i.e., it is always undefined). The `has_empty_codomain()` method will always be called before the methods below. However, if `has_empty_codomain()` returns `true`, none of the functions below will be called.

```
dimension_type max_in_codomain() const
```

returns the maximum value that belongs to the codomain of the partial function. The `max_in_-codomain()` method is called at most once.

```
bool maps(dimension_type i, dimension_type& j) const
```

Let f be the represented function and k be the value of `i`. If f is defined in k, then $f(k)$ is assigned to `j` and `true` is returned. If f is undefined in k, then `false` is returned. This method is called at most n times, where n is the dimension of the vector space enclosing the polyhedron.

The result is undefined if `pfunc` does not encode a partial function with the properties described in the specification of the mapping operator.

Definition at line 151 of file Polyhedron.templates.hh.

References Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_-Library::Constraint_System::clear(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Generator::coefficient(), con_sys, constraints_are_up_to_date(), Parma_-Polyhedra_Library::Generator::divisor(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_-Library::Generator_System::end(), gen_sys, generators(), generators_are_up_to_date(), Parma_-Polyhedra_Library::Matrix::has_no_rows(), has_pending_constraints(), Parma_Polyhedra_-Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator::LINE, marked_-empty(), Parma_Polyhedra_Library::not_a_dimension(), OK(), Parma_Polyhedra_Library::Linear_-System::permute_columns(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_-Library::Generator::RAY, remove_pending_to_obtain_generators(), set_zero_dim_univ(), space_dim,

swap(), throw_invalid_argument(), topology(), Parma_Polyhedra_Library::Generator::type(), and update_generators().

11.160.5.92 bool Parma_Polyhedra_Library::Polyhedron::marked_empty () const [inline, private]

Returns `true` if the polyhedron is known to be empty.

The return value `false` does not necessarily implies that `*this` is non-empty.

Definition at line 123 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_empty().

Referenced by add_congruence(), add_constraint(), add_generator(), add_recycled_constraints(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_poly_hull_assign_if_exact(), bounded_affine_image(), bounded_affine_preimage(), bounds(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), concatenate_assign(), constrains(), constraints(), contains(), contains_integer_point(), drop_some_non_integer_points(), fold_space_dimensions(), frequency(), generalized_affine_image(), generalized_affine_preimage(), generators(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), H79_widening_assign(), intersection_assign(), is_bounded(), is_empty(), is_included_in(), is_topologically_closed(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), max_min(), minimize(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), operator=(), poly_difference_assign(), poly_hull_assign(), process_pending(), process_pending_constraints(), process_pending_generators(), quick_equivalence_test(), refine_no_check(), refine_with_congruence(), refine_with_constraint(), refine_with_constraints(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), time_elapse_assign(), topological_closure_assign(), unconstrain(), update_constraints(), and update_generators().

11.160.5.93 bool Parma_Polyhedra_Library::Polyhedron::max_min (const Linear_Expression & *expr*, bool *maximize*, Coefficient & *ext_n*, Coefficient & *ext_d*, bool & *included*, Generator & *g*) const [private]

Maximizes or minimizes `expr` subject to `*this`.

Parameters

expr The linear expression to be maximized or minimized subject to `this`;

maximize `true` if maximization is what is wanted;

ext_n The numerator of the extremum value;

ext_d The denominator of the extremum value;

included `true` if and only if the extremum of `expr` can actually be reached in * this;

g When maximization or minimization succeeds, will be assigned a point or closure point where `expr` reaches the corresponding extremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded in the appropriate direction, `false` is returned and `ext_n`, `ext_d`, `included` and `g` are left untouched.

Definition at line 556 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::assign_r(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Linear_Expression::inhomogeneous_term(), Parma_Polyhedra_Library::Generator::is_closure_point(), Parma_Polyhedra_Library::Generator::is_line(), Parma_Polyhedra_Library::Generator::is_line_or_ray(), Parma_Polyhedra_Library::Generator::is_point(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, process_pending_constraints(), space_dim, Parma_Polyhedra_Library::Linear_Expression::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by maximize(), and minimize().

11.160.5.94 dimension_type Parma_Polyhedra_Library::Polyhedron::max_space_dimension () [inline, static]

Returns the maximum space dimension all kinds of Polyhedron can handle.

Definition at line 50 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Generator_System::max_space_dimension(), and Parma_Polyhedra_Library::Constraint_System::max_space_dimension().

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), concatenate_assign(), expand_space_dimension(), and Polyhedron().

11.160.5.95 bool Parma_Polyhedra_Library::Polyhedron::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*, Generator & *g*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from above in `*this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr The linear expression to be maximized subject to `*this`;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum `true` if and only if the supremum is also the maximum value;

g When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If *this is empty or expr is not bounded from above, false is returned and sup_n, sup_d, maximum and g are left untouched.

Definition at line 324 of file Polyhedron.inlines.hh.

References max_min().

11.160.5.96 bool Parma_Polyhedra_Library::Polyhedron::maximize (const Linear_Expression & *expr*, Coefficient & *sup_n*, Coefficient & *sup_d*, bool & *maximum*) const [inline]

Returns true if and only if *this is not empty and expr is bounded from above in *this, in which case the supremum value is computed.

Parameters

expr The linear expression to be maximized subject to *this;

sup_n The numerator of the supremum value;

sup_d The denominator of the supremum value;

maximum true if and only if the supremum is also the maximum value.

Exceptions

std::invalid_argument Thrown if expr and *this are dimension-incompatible.

If *this is empty or expr is not bounded from above, false is returned and sup_n, sup_d and maximum are left untouched.

Definition at line 316 of file Polyhedron.inlines.hh.

References max_min().

11.160.5.97 bool Parma_Polyhedra_Library::Polyhedron::minimize (bool *con_to_gen*, Linear_System & *source*, Linear_System & *dest*, Bit_Matrix & *sat*) [static, private]

Builds and simplifies constraints from generators (or vice versa).

Returns

true if the polyhedron is empty, false otherwise.

Parameters

con_to_gen true if source represents the constraints, false otherwise;

source The given system, which is not empty;

dest The system to build and minimize;

sat The saturation matrix.

dest is not const because it will be built (and then modified) during minimize(). Also, sat and source are not const because the former will be built during dest creation and the latter will maybe be sorted and modified by conversion() and simplify().

sat has the generators on its columns and the constraints on its rows if con_to_gen is true, otherwise it has the generators on its rows and the constraints on its columns.

Given source, this function builds (by means of conversion()) dest and then simplifies (invoking simplify()) source, erasing redundant rows. For the sequel we assume that source is the system of constraints and dest is the system of generators. This will simplify the description of the function; the dual case is similar.

Definition at line 70 of file minimize.cc.

References conversion(), Parma_Polyhedra_Library::Matrix::has_no_rows(), Parma_Polyhedra_Library::Linear_System::is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Linear_System::resize_no_copy(), Parma_Polyhedra_Library::Linear_System::set_index_first_pending_row(), Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality(), Parma_Polyhedra_Library::Linear_System::set_sorted(), simplify(), Parma_Polyhedra_Library::Linear_System::sort_rows(), Parma_Polyhedra_Library::Linear_System::topology(), and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

11.160.5.98 bool Parma_Polyhedra_Library::Polyhedron::minimize () const [private]

Applies (weak) minimization to both the constraints and generators.

Returns

false if and only if *this turns out to be an empty polyhedron.

Minimization is not attempted if the Status field already declares both systems to be minimized.

Definition at line 1039 of file Polyhedron_nonpublic.cc.

References constraints_are_minimized(), constraints_are_up_to_date(), generators_are_minimized(), generators_are_up_to_date(), has_something_pending(), marked_empty(), OK(), process_pending(), space_dim, update_constraints(), and update_generators().

Referenced by add_recycled_generators(), affine_image(), affine_preimage(), constrains(), generalized_affine_image(), is_empty(), is_universe(), minimized_constraints(), minimized_generators(), OK(), strongly_minimize_constraints(), strongly_minimize_generators(), update_constraints(), and update_generators().

11.160.5.99 bool Parma_Polyhedra_Library::Polyhedron::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*, Generator & *g*) const [inline]

Returns true if and only if *this is not empty and expr is bounded from below in *this, in which case the infimum value and a point where expr reaches it are computed.

Parameters

expr The linear expression to be minimized subject to *this;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value;

g When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

Definition at line 339 of file Polyhedron.inlines.hh.

References max_min().

11.160.5.100 bool Parma_Polyhedra_Library::Polyhedron::minimize (const Linear_Expression & *expr*, Coefficient & *inf_n*, Coefficient & *inf_d*, bool & *minimum*) const [inline]

Returns `true` if and only if `*this` is not empty and `expr` is bounded from below in `*this`, in which case the infimum value is computed.

Parameters

expr The linear expression to be minimized subject to `*this`;

inf_n The numerator of the infimum value;

inf_d The denominator of the infimum value;

minimum `true` if and only if the infimum is also the minimum value.

Exceptions

std::invalid_argument Thrown if `expr` and `*this` are dimension-incompatible.

If `*this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

Definition at line 331 of file Polyhedron.inlines.hh.

References max_min().

Referenced by BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::BHRZ03_-Certificate::BHRZ03_Certificate(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), H79_widening_assign(), is_included_-in(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), poly_difference_-assign(), and simplify_using_context_assign().

11.160.5.101 Congruence_System Parma_Polyhedra_Library::Polyhedron::minimized_-congruences () const [inline]

Returns a system of (equality) congruences satisfied by `*this`, with no redundant congruences and having the same affine dimension as `*this`.

Definition at line 362 of file Polyhedron.inlines.hh.

References minimized_constraints().

11.160.5.102 const PPL::Constraint_System & Parma_Polyhedra_Library::Polyhedron::minimized_constraints () const

Returns the system of constraints, with no redundant constraint.

Definition at line 119 of file Polyhedron_public.cc.

References constraints(), is_necessarily_closed(), minimize(), and strongly_minimize_constraints().

Referenced by affine_dimension(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), congruences(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), minimized_congruences(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimized_constraints(), and operator<<().

11.160.5.103 const PPL::Generator_System & Parma_Polyhedra_Library::Polyhedron::minimized_generators () const

Returns the system of generators, with no redundant generator.

Definition at line 184 of file Polyhedron_public.cc.

References generators(), is_necessarily_closed(), minimize(), and strongly_minimize_generators().

Referenced by Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), and Parma_Polyhedra_Library::BHRZ03_Certificate::compare().

11.160.5.104 Grid_Generator_System Parma_Polyhedra_Library::Polyhedron::minimized_grid_generators () const [inline]

Returns a universe system of grid generators.

Definition at line 367 of file Polyhedron.inlines.hh.

References grid_generators().

11.160.5.105 void Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints () const [private]

Sorts the matrix of constraints keeping status consistency.

It is assumed that constraints are up-to-date. If at least one of the saturation matrices is up-to-date, then `sat_g` is kept consistent with the sorted matrix of constraints. The method is declared `const` because reordering the constraints does not modify the polyhedron from a *logical* point of view.

Definition at line 914 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::check_sorted(), clear_sat_c_up_to_date(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::is_sorted(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), set_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_rows(), and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

Referenced by constraints(), is_universe(), and quick_equivalence_test().

11.160.5.106 void Parma_Polyhedra_Library::Polyhedron::obtain_sorted_constraints_with_sat_c () const [private]

Sorts the matrix of constraints and updates `sat_c`.

It is assumed that both constraints and generators are up-to-date and minimized. The method is declared `const` because reordering the constraints does not modify the polyhedron from a *logical* point of view.

Definition at line 970 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::check_sorted(), con_sys, constraints_are_minimized(), constraints_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::is_sorted(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), set_sat_c_up_to_date(), set_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Bit_Matrix::transpose_assign(), and update_sat_c().

Referenced by process_pending_constraints().

11.160.5.107 void Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators () const [private]

Sorts the matrix of generators keeping status consistency.

It is assumed that generators are up-to-date. If at least one of the saturation matrices is up-to-date, then `sat_c` is kept consistent with the sorted matrix of generators. The method is declared `const` because reordering the generators does not modify the polyhedron from a *logical* point of view.

Definition at line 942 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::check_sorted(), clear_sat_g_up_to_date(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::is_sorted(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), set_sat_c_up_to_date(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Linear_System::sort_rows(), and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

Referenced by generators(), and quick_equivalence_test().

11.160.5.108 void Parma_Polyhedra_Library::Polyhedron::obtain_sorted_generators_with_sat_g () const [private]

Sorts the matrix of generators and updates `sat_g`.

It is assumed that both constraints and generators are up-to-date and minimized. The method is declared `const` because reordering the generators does not modify the polyhedron from a *logical* point of view.

Definition at line 1005 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::check_sorted(), gen_sys, generators_are_up_to_date(), Parma_Polyhedra_Library::Linear_System::is_sorted(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), set_sat_c_up_to_date(), set_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_System::sort_and_remove_with_sat(), Parma_Polyhedra_Library::Bit_Matrix::transpose_assign(), and update_sat_g().

Referenced by process_pending_generators().

11.160.5.109 bool Parma_Polyhedra_Library::Polyhedron::OK (bool *check_not_empty* = `false`) const

Checks if all the invariants are satisfied.

Returns

`true` if and only if `*this` satisfies all the invariants and either `check_not_empty` is `false` or `*this` is not empty.

Parameters

check_not_empty `true` if and only if, in addition to checking the invariants, `*this` must be checked to be not empty.

The check is performed so as to intrude as little as possible. If the library has been compiled with run-time assertions enabled, error messages are written on `std::cerr` in case invariants are violated. This is useful for the purpose of debugging the library.

Definition at line 779 of file Polyhedron_public.cc.

References ascii_dump(), Parma_Polyhedra_Library::Constraint_System::ascii_dump(), Parma_Polyhedra_Library::Generator_System::ascii_dump(), con_sys, constraints_are_minimized(), constraints_are_up_to_date(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), gen_sys, generators_are_minimized(), generators_are_up_to_date(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Generator_System::has_points(), has_something_pending(), is_necessarily_closed(), marked_empty(), minimize(), Parma_Polyhedra_Library::Bit_Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Generator_System::num_rays(), Parma_Polyhedra_Library::Bit_Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Constraint_System::OK(), Parma_Polyhedra_Library::Generator_System::OK(), Parma_Polyhedra_Library::Polyhedron::Status::OK(), Parma_Polyhedra_Library::Bit_Matrix::OK(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Constraint_System::simplify(), Parma_Polyhedra_Library::Linear_System::sort_rows(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, Parma_Polyhedra_Library::Linear_System::strong_normalize(), topology(), Parma_Polyhedra_Library::Linear_System::topology(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by add_generator(), add_recycled_constraints(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(),

Parma_Polyhedra_Library::Polyhedron::Status::ascii_load(), BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_C_poly_hull_assign_if_exact(), bounded_affine_image(), bounded_affine_preimage(), Parma_Polyhedra_Library::C_Polyhedron::C_Polyhedron(), concatenate_assign(), drop_some_non_integer_points(), expand_space_dimension(), fold_space_dimensions(), generalized_affine_image(), generalized_affine_preimage(), H79_widening_assign(), intersection_assign(), is_included_in(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), minimize(), Parma_Polyhedra_Library::NNC_Polyhedron::NNC_Polyhedron(), poly_difference_assign(), poly_hull_assign(), Polyhedron(), process_pending_constraints(), process_pending_generators(), refine_no_check(), refine_with_constraints(), remove_higher_space_dimensions(), remove_pending_to_obtain_constraints(), remove_pending_to_obtain_generators(), remove_space_dimensions(), simplify_using_context_assign(), strongly_minimize_constraints(), strongly_minimize_generators(), time_elapse_assign(), topological_closure_assign(), and unconstrain().

11.160.5.110 PPL::Polyhedron & Parma_Polyhedra_Library::Polyhedron::operator= (const Polyhedron & *y*) [protected]

The assignment operator. (`*this` and `y` can be dimension-incompatible.).

Definition at line 311 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_System::assign_with_pending(), con_sys, constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), marked_empty(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), set_empty(), set_zero_dim_univ(), space_dim, status, and topology().

11.160.5.111 void Parma_Polyhedra_Library::Polyhedron::poly_difference_assign (const Polyhedron & *y*)

Assigns to `*this` the poly-difference of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 2469 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), constraints(), contains(), Parma_Polyhedra_Library::EMPTY, Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_tautological(), marked_empty(), minimize(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, OK(), poly_hull_assign(), refine_no_check(), relation_with(), set_empty(), space_dim, Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, throw_dimension_incompatible(), throw_topology_incompatible(), topology(), and Parma_Polyhedra_Library::Constraint::type().

Referenced by difference_assign().

11.160.5.112 void Parma_Polyhedra_Library::Polyhedron::poly_hull_assign (const Polyhedron & *y*)

Assigns to `*this` the poly-hull of `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 2403 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_rows(), Parma_Polyhedra_Library::Linear_System::add_rows(), can_have_something_pending(), clear_constraints_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), OK(), process_pending_constraints(), set_generators_pending(), space_dim, throw_dimension_incompatible(), throw_topology_incompatible(), topology(), and update_generators().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), fold_space_dimensions(), poly_difference_assign(), and upper_bound_assign().

11.160.5.113 void Parma_Polyhedra_Library::Polyhedron::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.160.5.114 bool Parma_Polyhedra_Library::Polyhedron::process_pending () const [inline, private]

Processes the pending rows of either description of the polyhedron and obtains a minimized polyhedron.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

It is assumed that the polyhedron does have some constraints or generators pending.

Definition at line 291 of file Polyhedron.inlines.hh.

References has_pending_constraints(), has_pending_generators(), has_something_pending(), marked_empty(), process_pending_constraints(), process_pending_generators(), and space_dim.

Referenced by contains_integer_point(), is_topologically_closed(), and minimize().

11.160.5.115 bool Parma_Polyhedra_Library::Polyhedron::process_pending_constraints () const [private]

Processes the pending constraints and obtains a minimized polyhedron.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

It is assumed that the polyhedron does have some pending constraints.

Definition at line 688 of file Polyhedron_nonpublic.cc.

References add_and_minimize(), clear_pending_constraints(), clear_sat_g_up_to_date(), con_sys, gen_sys, has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), obtain_sorted_constraints_with_sat_c(), OK(), sat_c, sat_c_is_up_to_date(), sat_g, set_empty(), set_sat_c_up_to_date(), Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), space_dim, and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

Referenced by add_generator(), add_recycled_generators(), bounds(), drop_some_non_integer_points(), frequency(), generators(), is_bounded(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), max_min(), poly_hull_assign(), process_pending(), relation_with(), remove_pending_to_obtain_generators(), time_elapse_assign(), topological_closure_assign(), and unconstrain().

11.160.5.116 void Parma_Polyhedra_Library::Polyhedron::process_pending_generators () const [private]

Processes the pending generators and obtains a minimized polyhedron.

It is assumed that the polyhedron does have some pending generators.

Definition at line 725 of file Polyhedron_nonpublic.cc.

References add_and_minimize(), clear_pending_generators(), clear_sat_c_up_to_date(), con_sys, gen_sys, has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), obtain_sorted_generators_with_sat_g(), OK(), sat_c, sat_g, sat_g_is_up_to_date(), set_sat_g_up_to_date(), Parma_Polyhedra_Library::Linear_System::sort_pending_and_remove_duplicates(), space_dim, and Parma_Polyhedra_Library::Bit_Matrix::transpose_assign().

Referenced by add_recycled_constraints(), concatenate_assign(), constrains(), constraints(), drop_some_non_integer_points(), H79_widening_assign(), intersection_assign(), is_included_in(), is_universe(), process_pending(), refine_no_check(), refine_with_constraints(), relation_with(), and remove_pending_to_obtain_constraints().

11.160.5.117 PPL::Polyhedron::Three_Valued_Boolean Parma_Polyhedra_Library::Polyhedron::quick_equivalence_test (const Polyhedron & *y*) const [private]

Polynomial but incomplete equivalence test between polyhedra.

Definition at line 334 of file Polyhedron_nonpublic.cc.

References con_sys, constraints_are_minimized(), gen_sys, generators_are_minimized(), has_something_pending(), marked_empty(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Matrix::num_rows(), obtain_sorted_constraints(), obtain_sorted_generators(), space_dim, topology(), TVB_DONT_KNOW, TVB_FALSE, and TVB_TRUE.

Referenced by contains().

11.160.5.118 void Parma_Polyhedra_Library::Polyhedron::refine_no_check (const Constraint & *c*) [private]

Uses a copy of constraint `c` to refine the system of constraints of `*this`.

Parameters

c The constraint to be added. If it is dimension-incompatible with `*this`, the behavior is undefined.

Definition at line 1355 of file Polyhedron_nonpublic.cc.

References can_have_something_pending(), clear_constraints_minimized(), clear_generators_up_-to_date(), con_sys, constraints_are_up_to_date(), has_pending_generators(), Parma_Polyhedra_-Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Constraint::is_-inconsistent(), is_necessarily_closed(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), marked_empty(), OK(), process_pending_generators(), set_constraints_pending(), set_empty(), space_-dim, Parma_Polyhedra_Library::Constraint::space_dimension(), and update_constraints().

Referenced by add_congruence(), add_constraint(), bounded_affine_image(), bounded_affine_preimage(), generalized_affine_image(), generalized_affine_preimage(), poly_difference_assign(), refine_with_-congruence(), and refine_with_constraint().

11.160.5.119 void Parma_Polyhedra_Library::Polyhedron::refine_with_congruence (const Congruence & *cg*)

Uses a copy of congruence `cg` to refine `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `cg` are dimension-incompatible.

Definition at line 1693 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_-Library::Congruence::is_equality(), Parma_Polyhedra_Library::Congruence::is_tautological(), Parma_-Polyhedra_Library::Checked::le, marked_empty(), Parma_Polyhedra_Library::NECESSARILY_-CLOSED, refine_no_check(), set_empty(), space_dim, Parma_Polyhedra_Library::Congruence::space_-dimension(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), and throw_dimension_-incompatible().

11.160.5.120 void Parma_Polyhedra_Library::Polyhedron::refine_with_congruences (const Congruence_System & *cgs*)

Uses a copy of the congruences in `cgs` to refine `*this`.

Parameters

cgs Contains the congruences used to refine the system of constraints of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `cgs` are dimension-incompatible.

Definition at line 1799 of file Polyhedron_public.cc.

References add_recycled_constraints(), Parma_Polyhedra_Library::Congruence_System::begin(), Parma_Polyhedra_Library::Congruence_System::end(), Parma_Polyhedra_Library::Constraint::EQUALITY, Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::NECESSARILY_CLOSED, set_empty(), space_dim, Parma_Polyhedra_Library::Congruence_System::space_dimension(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), and throw_dimension_incompatible().

11.160.5.121 void Parma_Polyhedra_Library::Polyhedron::refine_with_constraint (const Constraint & *c*)

Uses a copy of constraint `c` to refine `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 1683 of file Polyhedron_public.cc.

References marked_empty(), refine_no_check(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_dimension_incompatible().

11.160.5.122 void Parma_Polyhedra_Library::Polyhedron::refine_with_constraints (const Constraint_System & *cs*)

Uses a copy of the constraints in `cs` to refine `*this`.

Parameters

cs Contains the constraints used to refine the system of constraints of `*this`.

Exceptions

std::invalid_argument Thrown if `*this` and `cs` are dimension-incompatible.

Definition at line 1719 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Constraint_System::begin(), can_have_something_pending(), clear_constraints_minimized(), clear_generators_up_to_date(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_Polyhedra_Library::Matrix::has_no_rows(), has_pending_generators(), Parma_Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint_System::insert_pending(), Parma_Polyhedra_Library::Constraint::is_equality(), is_necessarily_closed(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_generators(), set_constraints_pending(), Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, throw_dimension_incompatible(), and update_constraints().

11.160.5.123 PPL::Poly_Con_Relation Parma_Polyhedra_Library::Polyhedron::relation_with (const Congruence & *cg*) const

Returns the relations holding between the polyhedron `*this` and the congruence `c`.

Exceptions

std::invalid_argument Thrown if `*this` and congruence `c` are dimension-incompatible.

Definition at line 276 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator_System::end(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Poly_Con_Relation::implies(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Congruence::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Congruence::is_inconsistent(), marked_empty(), Parma_Polyhedra_Library::Congruence::modulus(), PPL_DIRTY_TEMP_COEFFICIENT, process_pending_constraints(), relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), space_dim, Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_Polyhedra_Library::Poly_Con_Relation::strictly_intersects(), throw_dimension_incompatible(), and update_generators().

11.160.5.124 PPL::Poly_Gen_Relation Parma_Polyhedra_Library::Polyhedron::relation_with (const Generator & *g*) const

Returns the relations holding between the polyhedron `*this` and the generator `g`.

Exceptions

std::invalid_argument Thrown if `*this` and generator `g` are dimension-incompatible.

Definition at line 250 of file Polyhedron_public.cc.

References con_sys, constraints_are_up_to_date(), has_pending_generators(), marked_empty(), Parma_Polyhedra_Library::Poly_Gen_Relation::nothing(), process_pending_generators(), Parma_Polyhedra_Library::Constraint_System::satisfies_all_constraints(), space_dim, Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Poly_Gen_Relation::subsumes(), throw_dimension_incompatible(), and update_constraints().

11.160.5.125 PPL::Poly_Con_Relation Parma_Polyhedra_Library::Polyhedron::relation_with (const Constraint & *c*) const

Returns the relations holding between the polyhedron `*this` and the constraint `c`.

Exceptions

std::invalid_argument Thrown if `*this` and constraint `c` are dimension-incompatible.

Definition at line 210 of file Polyhedron_public.cc.

References gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Constraint::inhomogeneous_term(), Parma_Polyhedra_Library::Poly_Con_Relation::is_disjoint(), Parma_Polyhedra_Library::Constraint::is_equality(), Parma_Polyhedra_Library::Poly_Con_Relation::is_included(), Parma_Polyhedra_Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), marked_empty(), process_pending_constraints(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Poly_Con_Relation::saturates(), space_dim, Parma_Polyhedra_Library::Constraint::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), poly_difference_assign(), and relation_with().

11.160.5.126 void Parma_Polyhedra_Library::Polyhedron::remove_higher_space_dimensions (dimension_type *new_dimension*)

Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`.

Exceptions

std::invalid_argument Thrown if `new_dimensions` is greater than the space dimension of `*this`.

Definition at line 490 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Constraint_System::clear(), clear_constraints_up_to_date(), clear_generators_minimized(), con_sys, gen_sys, generators_are_up_to_date(), has_something_pending(), is_necessarily_closed(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), Parma_Polyhedra_Library::Generator_System::remove_invalid_lines_and_rays(), remove_pending_to_obtain_generators(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), set_zero_dim_univ(), space_dim, Parma_Polyhedra_Library::Matrix::swap_columns(), throw_dimension_incompatible(), and update_generators().

Referenced by bounded_affine_image(), bounded_affine_preimage(), generalized_affine_image(), and generalized_affine_preimage().

11.160.5.127 void Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_constraints () const `[private]`

Lazily integrates the pending descriptions of the polyhedron to obtain a constraint system without pending rows.

It is assumed that the polyhedron does have some constraints or generators pending.

Definition at line 756 of file Polyhedron_nonpublic.cc.

References clear_constraints_minimized(), clear_generators_up_to_date(), clear_pending_constraints(), con_sys, has_pending_constraints(), has_pending_generators(), has_something_pending(), OK(), process_pending_generators(), Parma_Polyhedra_Library::Linear_System::set_sorted(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by affine_preimage().

11.160.5.128 bool Parma_Polyhedra_Library::Polyhedron::remove_pending_to_obtain_-generators () const [private]

Lazily integrates the pending descriptions of the polyhedron to obtain a generator system without pending rows.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

It is assumed that the polyhedron does have some constraints or generators pending.

Definition at line 780 of file Polyhedron_nonpublic.cc.

References clear_constraints_up_to_date(), clear_generators_minimized(), clear_pending_generators(), gen_sys, has_pending_constraints(), has_pending_generators(), has_something_pending(), OK(), process_pending_constraints(), Parma_Polyhedra_Library::Linear_System::set_sorted(), and Parma_-Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by affine_image(), map_space_dimensions(), remove_higher_space_dimensions(), and remove_space_dimensions().

11.160.5.129 void Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions (const Variables_Set & *vars*)

Removes all the specified dimensions from the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be removed.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 417 of file Polyhedron_chdims.cc.

References Parma_Polyhedra_Library::Constraint_System::clear(), clear_constraints_up_to_date(), clear_generators_minimized(), con_sys, gen_sys, generators_are_up_to_date(), has_something_pending(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_columns(), OK(), Parma_Polyhedra_-Library::Generator_System::remove_invalid_lines_and_rays(), remove_pending_to_obtain_generators(), Parma_Polyhedra_Library::Linear_System::remove_trailing_columns(), set_zero_dim_univ(), space_-dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by fold_space_dimensions().

11.160.5.130 bool Parma_Polyhedra_Library::Polyhedron::sat_c_is_up_to_date () const [inline, private]

Returns `true` if the saturation matrix `sat_c` is up-to-date.

Definition at line 148 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_sat_c_up_to_date().

Referenced by add_space_dimensions_and_embed(), can_have_something_pending(), concatenate_assign(), obtain_sorted_constraints(), obtain_sorted_constraints_with_sat_c(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), OK(), operator=(), Polyhedron(), process_pending_constraints(), strongly_minimize_constraints(), strongly_minimize_generators(), and update_sat_c().

11.160.5.131 bool Parma_Polyhedra_Library::Polyhedron::sat_g_is_up_to_date () const [inline, private]

Returns `true` if the saturation matrix `sat_g` is up-to-date.

Definition at line 153 of file Polyhedron.inlines.hh.

References status, and Parma_Polyhedra_Library::Polyhedron::Status::test_sat_g_up_to_date().

Referenced by add_space_dimensions_and_project(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), can_have_something_pending(), obtain_sorted_constraints(), obtain_sorted_constraints_with_sat_c(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), OK(), operator=(), Polyhedron(), process_pending_generators(), select_H79_constraints(), strongly_minimize_constraints(), strongly_minimize_generators(), and update_sat_g().

11.160.5.132 void Parma_Polyhedra_Library::Polyhedron::select_CH78_constraints (const Polyhedron & *y*, Constraint_System & *cs_selected*) const [private]

Copies to `cs_selection` the constraints of `y` corresponding to the definition of the CH78-widening of `*this` and `y`.

Definition at line 40 of file Polyhedron_widenings.cc.

References con_sys, constraints_are_minimized(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), has_something_pending(), Parma_Polyhedra_Library::Constraint_System::insert(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), space_dim, Parma_Polyhedra_Library::Linear_System::topology(), and topology().

Referenced by H79_widening_assign().

11.160.5.133 void Parma_Polyhedra_Library::Polyhedron::select_H79_constraints (const Polyhedron & *y*, Constraint_System & *cs_selected*, Constraint_System & *cs_not_selected*) const [private]

Splits the constraints of 'x' into two subsets, depending on whether or not they are selected to compute the H79-widening of `*this` and `y`.

Definition at line 67 of file Polyhedron_widenings.cc.

References Parma_Polyhedra_Library::Bit_Row::clear(), con_sys, constraints_are_minimized(), constraints_are_up_to_date(), gen_sys, generators_are_up_to_date(), has_pending_generators(), has_-

something_pending(), Parma_Polyhedra_Library::Constraint_System::insert(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_strict_inequality(), marked_empty(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Bit_Matrix::rows_erase_to_end(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Bit_Matrix::sort_rows(), Parma_Polyhedra_Library::Bit_Matrix::sorted_contains(), space_dim, strongly_minimize_constraints(), swap(), Parma_Polyhedra_Library::Linear_System::topology(), topology(), update_generators(), and update_sat_g().

Referenced by BHRZ03_widening_assign(), and H79_widening_assign().

11.160.5.134 void Parma_Polyhedra_Library::Polyhedron::set_constraints_minimized () [inline, private]

Sets `status` to express that constraints are minimized.

Definition at line 202 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_c_minimized(), set_constraints_up_to_date(), and status.

Referenced by Polyhedron(), update_constraints(), and update_generators().

11.160.5.135 void Parma_Polyhedra_Library::Polyhedron::set_constraints_pending () [inline, private]

Sets `status` to express that constraints are pending.

Definition at line 214 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_c_pending(), and status.

Referenced by add_recycled_constraints(), concatenate_assign(), intersection_assign(), refine_no_check(), and refine_with_constraints().

11.160.5.136 void Parma_Polyhedra_Library::Polyhedron::set_constraints_up_to_date () [inline, private]

Sets `status` to express that constraints are up-to-date.

Definition at line 192 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_c_up_to_date(), and status.

Referenced by Polyhedron(), and set_constraints_minimized().

11.160.5.137 void Parma_Polyhedra_Library::Polyhedron::set_empty () [private]

Sets `status` to express that the polyhedron is empty, clearing all corresponding matrices.

Definition at line 678 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Bit_Matrix::clear(), Parma_Polyhedra_Library::Generator_System::clear(), Parma_Polyhedra_Library::Constraint_System::clear(), con_sys, gen_sys, sat_c, sat_g, Parma_Polyhedra_Library::Polyhedron::Status::set_empty(), and status.

Referenced by add_congruence(), add_congruences(), add_constraint(), add_recycled_constraints(), Parma_Polyhedra_Library::Polyhedron::Status::ascii_load(), concatenate_assign(), drop_some_non_integer_points(), intersection_assign(), operator=(), poly_difference_assign(), Polyhedron(), process_pending_constraints(), refine_no_check(), refine_with_congruence(), refine_with_congruences(), time_elapse_assign(), and update_generators().

11.160.5.138 void Parma_Polyhedra_Library::Polyhedron::set_generators_minimized () [inline, private]

Sets `status` to express that generators are minimized.

Definition at line 208 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_g_minimized(), set_generators_up_to_date(), and status.

Referenced by add_generator(), add_space_dimensions_and_project(), update_constraints(), and update_generators().

11.160.5.139 void Parma_Polyhedra_Library::Polyhedron::set_generators_pending () [inline, private]

Sets `status` to express that generators are pending.

Definition at line 219 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_g_pending(), and status.

Referenced by add_generator(), add_recycled_generators(), poly_hull_assign(), time_elapse_assign(), topological_closure_assign(), and unconstrain().

11.160.5.140 void Parma_Polyhedra_Library::Polyhedron::set_generators_up_to_date () [inline, private]

Sets `status` to express that generators are up-to-date.

Definition at line 197 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_g_up_to_date(), and status.

Referenced by add_recycled_generators(), Polyhedron(), and set_generators_minimized().

11.160.5.141 void Parma_Polyhedra_Library::Polyhedron::set_sat_c_up_to_date () [inline, private]

Sets `status` to express that `sat_c` is up-to-date.

Definition at line 224 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_sat_c_up_to_date(), and status.

Referenced by Parma_Polyhedra_Library::Polyhedron::Status::ascii_load(), concatenate_assign(), obtain_sorted_constraints_with_sat_c(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), process_pending_constraints(), update_constraints(), and update_sat_c().

11.160.5.142 void Parma_Polyhedra_Library::Polyhedron::set_sat_g_up_to_date () [inline, private]

Sets `status` to express that `sat_g` is up-to-date.

Definition at line 229 of file Polyhedron.inlines.hh.

References Parma_Polyhedra_Library::Polyhedron::Status::set_sat_g_up_to_date(), and status.

Referenced by Parma_Polyhedra_Library::Polyhedron::Status::ascii_load(), obtain_sorted_constraints(), obtain_sorted_constraints_with_sat_c(), obtain_sorted_generators_with_sat_g(), process_pending_generators(), update_generators(), and update_sat_g().

11.160.5.143 void Parma_Polyhedra_Library::Polyhedron::set_zero_dim_univ () [private]

Sets `status` to express that the polyhedron is the universe 0-dimension vector space, clearing all corresponding matrices.

Definition at line 670 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator_System::clear(), Parma_Polyhedra_Library::Constraint_System::clear(), con_sys, gen_sys, Parma_Polyhedra_Library::Polyhedron::Status::set_zero_dim_univ(), space_dim, and status.

Referenced by add_generator(), add_recycled_generators(), Parma_Polyhedra_Library::Polyhedron::Status::ascii_load(), map_space_dimensions(), operator=(), Polyhedron(), remove_higher_space_dimensions(), remove_space_dimensions(), and simplify_using_context_assign().

11.160.5.144 Constraint_System Parma_Polyhedra_Library::Polyhedron::simplified_constraints () const [inline, private]

If constraints are up-to-date, obtain a simplified copy of them.

Definition at line 346 of file Polyhedron.inlines.hh.

References con_sys, constraints_are_minimized(), constraints_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Linear_System::num_pending_rows(), Parma_Polyhedra_Library::Constraint_System::simplify(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by Parma_Polyhedra_Library::Box< ITV >::Box().

11.160.5.145 PPL::dimension_type Parma_Polyhedra_Library::Polyhedron::simplify (Linear_System & *sys*, Bit_Matrix & *sat*) [static, private]

Uses Gauss' elimination method to simplify the result of `conversion()`.

Returns

The rank of `sys`.

Parameters

sys The system to simplify: it will be modified;

sat The saturation matrix corresponding to `sys`.

`sys` may be modified by swapping some of its rows and by possibly removing some of them, if they turn out to be redundant.

If `sys` is a system of constraints, then the rows of `sat` are indexed by constraints and its columns are indexed by generators; otherwise, if `sys` is a system of generators, then the rows of `sat` are indexed by generators and its columns by constraints.

Given a system of constraints or a system of generators, this function simplifies it using Gauss' elimination method (to remove redundant equalities/lines), deleting redundant inequalities/rays/points and making back-substitution. The explanation that follows assumes that `sys` is a system of constraints. For the case when `sys` is a system of generators, a similar explanation can be obtain by applying duality.

The explanation relies on the notion of *redundancy*. (See the Introduction.)

First we make some observations that can help the reader in understanding the function:

Proposition: An inequality that is saturated by all the generators can be transformed to an equality.

In fact, by combining any number of generators that saturate the constraints, we obtain a generator that saturates the constraints too:

$$\langle \boldsymbol{c}, \boldsymbol{r}_1 \rangle = 0 \wedge \langle \boldsymbol{c}, \boldsymbol{r}_2 \rangle = 0 \Rightarrow \langle \boldsymbol{c}, (\lambda_1 \boldsymbol{r}_1 + \lambda_2 \boldsymbol{r}_2) \rangle = \lambda_1 \langle \boldsymbol{c}, \boldsymbol{r}_1 \rangle + \lambda_2 \langle \boldsymbol{c}, \boldsymbol{r}_2 \rangle = 0,$$

where λ_1, λ_2 can be any real number.

Definition at line 84 of file simplify.cc.

References Parma_Polyhedra_Library::Linear_System::back_substitute(), Parma_Polyhedra_Library::compute_capacity(), Parma_Polyhedra_Library::Matrix::erase_to_end(), Parma_Polyhedra_Library::Linear_System::gauss(), Parma_Polyhedra_Library::Bit_Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Linear_System::OK(), Parma_Polyhedra_Library::Bit_Matrix::rows_erase_to_end(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::Linear_System::sign_normalize(), simplify_num_saturators_p, simplify_num_saturators_size, swap(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by add_and_minimize(), and minimize().

11.160.5.146 bool Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign (const Polyhedron & *y*)

Assigns to `*this` a meet-preserving simplification of `*this` with respect to `y`. If `false` is returned, then the intersection is empty.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 2055 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::MIP_Problem::add_constraint(), Parma_Polyhedra_Library::MIP_Problem::add_constraints(), add_constraints(), add_recycled_constraints(), Parma_Polyhedra_Library::MIP_Problem::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Generator_System::begin(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, con_sys, constraints_are_minimized(), constraints_are_up_to_date(), Parma_Polyhedra_Library::Bit_Row::empty(), Parma_Polyhedra_Library::Generator_System::end(), Parma_Polyhedra_Library::Constraint::EQUALITY, gen_sys, generators_are_minimized(), has_pending_generators(), has_something_pending(), Parma_Polyhedra_Library::Constraint_System::insert(), is_empty(), Parma_Polyhedra_Library::Constraint::is_inequality(), Parma_Polyhedra_Library::Generator::is_line(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_Polyhedra_Library::Checked::le, Parma_Polyhedra_Library::Generator::LINE, minimize(), Parma_Polyhedra_Library::Constraint::NONSTRICT_INEQUALITY, Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, Parma_Polyhedra_Library::Generator_System::satisfied_by_all_generators(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Linear_Row::set_is_line_or_equality(), set_zero_dim_univ(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Linear_Row::sign_normalize(), Parma_Polyhedra_Library::MIP_Problem::solve(), space_dim, status, Parma_Polyhedra_Library::Constraint::STRICT_INEQUALITY, Parma_Polyhedra_Library::Constraint_System::swap(), swap(), throw_dimension_incompatible(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::Generator::type(), Parma_Polyhedra_Library::Constraint::type(), Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM, and Parma_Polyhedra_Library::UNIVERSE.

11.160.5.147 dimension_type Parma_Polyhedra_Library::Polyhedron::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

Definition at line 40 of file Polyhedron.inlines.hh.

References space_dim.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BHRZ03_Certificate::BHRZ03_Certificate(), Parma_Polyhedra_Library::Box< ITV >::Box(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::check_containment(), Parma_Polyhedra_Library::H79_Certificate::compare(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), expand_space_dimension(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::H79_Certificate::H79_Certificate(), hash_code(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::NNC_Polyhedron::poly_hull_assign_if_exact(), Parma_Polyhedra_Library::C_Polyhedron::poly_hull_assign_if_exact(), and throw_dimension_incompatible().

11.160.5.148 bool Parma_Polyhedra_Library::Polyhedron::strictly_contains (const Polyhedron & *y*) const [inline]

Returns `true` if and only if `*this` strictly contains `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 383 of file Polyhedron.inlines.hh.

References contains().

11.160.5.149 bool Parma_Polyhedra_Library::Polyhedron::strongly_minimize_constraints () const [private]

Applies strong minimization to the constraints of an NNC polyhedron.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

Definition at line 1079 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::MIP_Problem::add_constraints(), Parma_Polyhedra_Library::MIP_Problem::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Bit_Row::clear(), clear_generators_up_to_date(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, con_sys, Parma_Polyhedra_Library::Constraint::epsilon_leq_one(), Parma_Polyhedra_Library::Matrix::erase_to_end(), gen_sys, Parma_Polyhedra_Library::Constraint_System::insert(), is_necessarily_closed(), Parma_Polyhedra_Library::MAXIMIZATION, minimize(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Generator::POINT, Parma_Polyhedra_Library::Generator::RAY, sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Linear_System::set_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::set_not_necessarily_closed(), Parma_Polyhedra_Library::MIP_Problem::set_objective_function(), Parma_Polyhedra_Library::MIP_Problem::set_optimization_mode(), Parma_Polyhedra_Library::Linear_System::set_sorted(), Parma_Polyhedra_Library::MIP_Problem::solve(), space_dim, Parma_Polyhedra_Library::Constraint_System::space_dimension(), status, swap(), Parma_Polyhedra_Library::Bit_Matrix::transpose_assign(), Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM, and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by is_topologically_closed(), minimized_constraints(), and select_H79_constraints().

11.160.5.150 bool Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators () const [private]

Applies strong minimization to the generators of an NNC polyhedron.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

Definition at line 1257 of file Polyhedron_nonpublic.cc.

References clear_constraints_up_to_date(), con_sys, Parma_Polyhedra_Library::Matrix::erase_to_end(), gen_sys, is_necessarily_closed(), minimize(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Constraint_System::num_equalities(), Parma_Polyhedra_Library::Generator_System::num_lines(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), sat_c, sat_c_is_up_to_date(), sat_g, sat_g_is_up_to_date(), Parma_Polyhedra_Library::Bit_Row::set(), Parma_Polyhedra_Library::Linear_System::set_sorted(), space_dim, swap(), Parma_Polyhedra_Library::Bit_Matrix::transpose_assign(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

Referenced by minimized_generators().

11.160.5.151 void Parma_Polyhedra_Library::Polyhedron::swap (Polyhedron & *y*) [inline]

Swaps `*this` with polyhedron `y`. (`*this` and `y` can be dimension-incompatible.).

Exceptions

std::invalid_argument Thrown if `x` and `y` are topology-incompatible.

Definition at line 100 of file Polyhedron.inlines.hh.

References con_sys, gen_sys, sat_c, sat_g, space_dim, status, throw_topology_incompatible(), and topology().

Referenced by add_recycled_generators(), add_space_dimensions(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_NNC_poly_hull_assign_if_exact(), concatenate_assign(), constraints(), conversion(), generators(), H79_widening_assign(), map_space_dimensions(), Parma_Polyhedra_Library::NNC_Polyhedron::operator=(), Parma_Polyhedra_Library::C_Polyhedron::operator=(), Polyhedron(), select_H79_constraints(), simplify(), simplify_using_context_assign(), strongly_minimize_constraints(), strongly_minimize_generators(), swap(), and time_elapse_assign().

11.160.5.152 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, dimension_type *required_space_dim*) const [protected]

Definition at line 2396 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed(), and space_dimension().

11.160.5.153 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *var_name*, Variable *var*) const [protected]

Definition at line 2379 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed(), Parma_Polyhedra_Library::Variable::space_dimension(), and space_dimension().

11.160.5.154 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *cgs_name*, const Congruence_System & *cgs*) const [protected]

Definition at line 2372 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Congruence_System::space_dimension(), and throw_dimension_-incompatible().

11.160.5.155 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *gs_name*, const Generator_System & *gs*) const [protected]

Definition at line 2365 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator_System::space_dimension(), and throw_dimension_-incompatible().

11.160.5.156 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *cs_name*, const Constraint_System & *cs*) const [protected]

Definition at line 2358 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint_System::space_dimension(), and throw_dimension_-incompatible().

11.160.5.157 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *cg_name*, const Congruence & *cg*) const [protected]

Definition at line 2351 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Congruence::space_dimension(), and throw_dimension_-incompatible().

11.160.5.158 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *g_name*, const Generator & *g*) const [protected]

Definition at line 2344 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Generator::space_dimension(), and throw_dimension_-incompatible().

11.160.5.159 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *c_name*, const Constraint & *c*) const [protected]

Definition at line 2337 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Constraint::space_dimension(), and throw_dimension_incompatible().

11.160.5.160 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *e_name*, const Linear_Expression & *e*) const [protected]

Definition at line 2330 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Linear_Expression::space_dimension(), and throw_dimension_incompatible().

11.160.5.161 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *ph_name*, const Polyhedron & *ph*) const [protected]

Definition at line 2323 of file Polyhedron_nonpublic.cc.

References space_dimension(), and throw_dimension_incompatible().

11.160.5.162 void Parma_Polyhedra_Library::Polyhedron::throw_dimension_incompatible (const char * *method*, const char * *other_name*, dimension_type *other_dim*) const [protected]

Definition at line 2310 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed(), and space_dimension().

Referenced by add_congruence(), add_congruences(), add_constraint(), add_generator(), add_recycled_constraints(), add_recycled_generators(), affine_image(), affine_preimage(), BHRZ03_widening_assign(), bounded_affine_image(), bounded_affine_preimage(), bounds(), constrains(), contains(), expand_space_dimension(), fold_space_dimensions(), frequency(), generalized_affine_image(), generalized_affine_preimage(), H79_widening_assign(), intersection_assign(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), max_min(), poly_difference_assign(), poly_hull_assign(), Parma_Polyhedra_Library::NNC_Polyhedron::poly_hull_assign_if_exact(), Parma_Polyhedra_Library::C_Polyhedron::poly_hull_assign_if_exact(), refine_with_congruence(), refine_with_congruences(), refine_with_constraint(), refine_with_constraints(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), simplify_using_context_assign(), throw_dimension_incompatible(), time_elapse_assign(), and unconstrain().

11.160.5.163 void Parma_Polyhedra_Library::Polyhedron::throw_invalid_argument (const char * *method*, const char * *reason*) const [protected]

Definition at line 2233 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

Referenced by add_congruence(), add_congruences(), affine_image(), affine_preimage(), bounded_affine_image(), bounded_affine_preimage(), fold_space_dimensions(), generalized_affine_image(), generalized_affine_preimage(), and map_space_dimensions().

11.160.5.164 void Parma_Polyhedra_Library::Polyhedron::throw_invalid_generator (const char * *method*, const char * *g_name*) const [protected]

Definition at line 2426 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

Referenced by add_generator().

11.160.5.165 void Parma_Polyhedra_Library::Polyhedron::throw_invalid_generators (const char * *method*, const char * *gs_name*) const [protected]

Definition at line 2441 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

Referenced by add_recycled_generators(), and Polyhedron().

11.160.5.166 void Parma_Polyhedra_Library::Polyhedron::throw_runtime_error (const char * *method*) const [protected]

Definition at line 2221 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

Referenced by add_generator().

11.160.5.167 void Parma_Polyhedra_Library::Polyhedron::throw_space_dimension_overflow (Topology *topol*, const char * *method*, const char * *reason*) [static, protected]

Definition at line 2411 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::NECESSARILY_CLOSED.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), concatenate_assign(), and expand_space_dimension().

11.160.5.168 void Parma_Polyhedra_Library::Polyhedron::throw_topology_incompatible (const char * *method*, const char * *gs_name*, const Generator_System & *gs*) const [protected]

Definition at line 2300 of file Polyhedron_nonpublic.cc.

11.160.5.169 void Parma_Polyhedra_Library::Polyhedron::throw_topology_incompatible (const char * *method*, const char * *cs_name*, const Constraint_System & *cs*) const [protected]

Definition at line 2289 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

11.160.5.170 void Parma_Polyhedra_Library::Polyhedron::throw_topology_incompatible (const char * *method*, const char * *g_name*, const Generator & *g*) const [protected]

Definition at line 2278 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

11.160.5.171 void Parma_Polyhedra_Library::Polyhedron::throw_topology_incompatible (const char * *method*, const char * *c_name*, const Constraint & *c*) const [protected]

Definition at line 2267 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

11.160.5.172 void Parma_Polyhedra_Library::Polyhedron::throw_topology_incompatible (const char * *method*, const char * *ph_name*, const Polyhedron & *ph*) const [protected]

Definition at line 2247 of file Polyhedron_nonpublic.cc.

References is_necessarily_closed().

Referenced by add_constraint(), add_generator(), add_recycled_constraints(), add_recycled_generators(), BHRZ03_widening_assign(), concatenate_assign(), contains(), H79_widening_assign(), intersection_assign(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), poly_difference_assign(), poly_hull_assign(), Polyhedron(), simplify_using_context_assign(), swap(), and time_elapse_assign().

11.160.5.173 void Parma_Polyhedra_Library::Polyhedron::time_elapse_assign (const Polyhedron & *y*)

Assigns to `*this` the result of computing the time-elapse between `*this` and `y`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are topology-incompatible or dimension-incompatible.

Definition at line 3341 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Linear_System::add_pending_rows(), Parma_Polyhedra_Library::Linear_Row::all_homogeneous_terms_are_zero(), can_have_something_pending(), clear_constraints_up_to_date(), clear_generators_minimized(), Parma_Polyhedra_Library::Generator::CLOSURE_POINT, Parma_Polyhedra_Library::Matrix::erase_to_end(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), is_necessarily_closed(), Parma_Polyhedra_Library::Linear_System::is_sorted(), marked_empty(), Parma_Polyhedra_Library::Linear_System::merge_rows_assign(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), Parma_Polyhedra_Library::Generator::POINT, process_pending_constraints(), set_empty(), set_generators_pending(), Parma_Polyhedra_Library::Linear_System::sort_rows(), space_dim, swap(), throw_dimension_incompatible(), throw_topology_incompatible(), topology(), Parma_Polyhedra_Library::Linear_System::unset_pending_rows(), and update_generators().

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::time_elapse_assign(), and Parma_Polyhedra_Library::BD_Shape< T >::time_elapse_assign().

11.160.5.174 void Parma_Polyhedra_Library::Polyhedron::topological_closure_assign ()

Assigns to `*this` its topological closure.

Definition at line 3549 of file Polyhedron_public.cc.

References Parma_Polyhedra_Library::Generator_System::add_corresponding_points(), can_have_something_pending(), clear_constraints_minimized(), clear_constraints_up_to_date(), clear_generators_minimized(), clear_generators_up_to_date(), con_sys, constraints_are_up_to_date(), Parma_Polyhedra_Library::Constraint::epsilon_leq_one(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), has_pending_generators(), Parma_Polyhedra_Library::Constraint_System::insert(), is_necessarily_closed(), Parma_Polyhedra_Library::Constraint::is_tautological(), marked_empty(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::Matrix::num_rows(), OK(), process_pending_constraints(), set_generators_pending(), Parma_Polyhedra_Library::Linear_System::set_sorted(), and Parma_Polyhedra_Library::Linear_System::unset_pending_rows().

11.160.5.175 Topology Parma_Polyhedra_Library::Polyhedron::topology () const [inline, private]

Returns the topological kind of the polyhedron.

Definition at line 62 of file Polyhedron.inlines.hh.

References con_sys, and Parma_Polyhedra_Library::Linear_System::topology().

Referenced by add_generator(), add_recycled_constraints(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_poly_hull_assign_if_exact(), concatenate_assign(), constraints(), contains(), expand_space_dimension(), generators(), H79_widening_assign(), intersection_assign(), is_included_in(), map_space_dimensions(), OK(), operator=(), poly_difference_assign(), poly_hull_assign(), Polyhedron(), quick_equivalence_test(), select_CH78_constraints(), select_H79_constraints(), simplify_using_context_assign(), swap(), and time_elapse_assign().

11.160.5.176 memory_size_type Parma_Polyhedra_Library::Polyhedron::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 35 of file Polyhedron.inlines.hh.

References external_memory_in_bytes().

11.160.5.177 void Parma_Polyhedra_Library::Polyhedron::unconstrain (const Variables_Set & *vars*)

Computes the cylindrification of `*this` with respect to the set of space dimensions `vars`, assigning the result to `*this`.

Parameters

vars The set of space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `*this` is dimension-incompatible with one of the Variable objects contained in `vars`.

Definition at line 1859 of file Polyhedron_public.cc.

References can_have_something_pending(), clear_constraints_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Generator::line(), marked_empty(), OK(), process_pending_constraints(), set_generators_pending(), space_dim, Parma_Polyhedra_Library::Variables_Set::space_dimension(), throw_dimension_incompatible(), and update_generators().

11.160.5.178 void Parma_Polyhedra_Library::Polyhedron::unconstrain (Variable *var*)

Computes the cylindrification of `*this` with respect to space dimension `var`, assigning the result to `*this`.

Parameters

var The space dimension that will be unconstrained.

Exceptions

std::invalid_argument Thrown if `var` is not a space dimension of `*this`.

Definition at line 1829 of file Polyhedron_public.cc.

References can_have_something_pending(), clear_constraints_up_to_date(), clear_generators_minimized(), gen_sys, generators_are_up_to_date(), has_pending_constraints(), Parma_Polyhedra_Library::Generator_System::insert(), Parma_Polyhedra_Library::Generator_System::insert_pending(), Parma_Polyhedra_Library::Generator::line(), marked_empty(), OK(), process_pending_constraints(), set_generators_pending(), space_dim, Parma_Polyhedra_Library::Variable::space_dimension(), throw_dimension_incompatible(), and update_generators().

Referenced by bounded_affine_preimage(), and generalized_affine_preimage().

11.160.5.179 void Parma_Polyhedra_Library::Polyhedron::update_constraints () const [private]

Updates constraints starting from generators and minimizes them.

The resulting system of constraints is only partially sorted: the equalities are in the upper part of the matrix, while the inequalities in the lower part.

Definition at line 805 of file Polyhedron_nonpublic.cc.

References clear_sat_g_up_to_date(), con_sys, gen_sys, generators_are_up_to_date(), has_something_pending(), marked_empty(), minimize(), sat_c, set_constraints_minimized(), set_generators_minimized(), set_sat_c_up_to_date(), and space_dim.

Referenced by add_recycled_constraints(), concatenate_assign(), constrains(), constraints(), drop_some_non_integer_points(), H79_widening_assign(), intersection_assign(), is_included_in(), minimize(), refine_no_check(), refine_with_constraints(), and relation_with().

11.160.5.180 bool Parma_Polyhedra_Library::Polyhedron::update_generators () const [private]

Updates generators starting from constraints and minimizes them.

Returns

`false` if and only if `*this` turns out to be an empty polyhedron.

The resulting system of generators is only partially sorted: the lines are in the upper part of the matrix, while rays and points are in the lower part. It is illegal to call this method when the Status field already declares the polyhedron to be empty.

Definition at line 824 of file Polyhedron_nonpublic.cc.

References clear_sat_c_up_to_date(), con_sys, constraints_are_up_to_date(), gen_sys, has_something_pending(), marked_empty(), minimize(), sat_g, set_constraints_minimized(), set_empty(), set_generators_minimized(), set_sat_g_up_to_date(), and space_dim.

Referenced by add_generator(), bounds(), frequency(), generators(), is_bounded(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), max_min(), minimize(), poly_hull_assign(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_H79_constraints(), time_elapse_assign(), and unconstrain().

11.160.5.181 void Parma_Polyhedra_Library::Polyhedron::update_sat_c () const [private]

Updates `sat_c` using the updated constraints and generators.

It is assumed that constraints and generators are up-to-date and minimized and that the Status field does not already flag `sat_c` to be up-to-date. The values of the saturation matrix are computed as follows:

$$\begin{cases} sat_c[i][j] = 0, & \text{if } G[i] \cdot C^{\mathrm{T}}[j] = 0; \\ sat_c[i][j] = 1, & \text{if } G[i] \cdot C^{\mathrm{T}}[j] > 0. \end{cases}$$

Definition at line 850 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Bit_Matrix::clear(), con_sys, constraints_are_minimized(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Bit_Matrix::resize(), sat_c, sat_c_is_up_to_date(), set_sat_c_up_to_date(), and Parma_Polyhedra_Library::Scalar_Products::sign().

Referenced by add_space_dimensions_and_embed(), and obtain_sorted_constraints_with_sat_c().

11.160.5.182 void Parma_Polyhedra_Library::Polyhedron::update_sat_g () const [private]

Updates `sat_g` using the updated constraints and generators.

It is assumed that constraints and generators are up-to-date and minimized and that the Status field does not already flag `sat_g` to be up-to-date. The values of the saturation matrix are computed as follows:

$$\begin{cases} sat_g[i][j] = 0, & \text{if } C[i] \cdot G^{\mathrm{T}}[j] = 0; \\ sat_g[i][j] = 1, & \text{if } C[i] \cdot G^{\mathrm{T}}[j] > 0. \end{cases}$$

Definition at line 882 of file Polyhedron_nonpublic.cc.

References Parma_Polyhedra_Library::Bit_Matrix::clear(), con_sys, constraints_are_minimized(), Parma_Polyhedra_Library::Linear_System::first_pending_row(), gen_sys, generators_are_minimized(), Parma_Polyhedra_Library::Bit_Matrix::resize(), sat_g, sat_g_is_up_to_date(), set_sat_g_up_to_date(), and Parma_Polyhedra_Library::Scalar_Products::sign().

Referenced by add_space_dimensions_and_project(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_-NNC_poly_hull_assign_if_exact(), obtain_sorted_generators_with_sat_g(), and select_H79_constraints().

11.160.5.183 void Parma_Polyhedra_Library::Polyhedron::upper_bound_assign (const Polyhedron & *y*) [inline]

Same as poly_hull_assign(y).

Definition at line 81 of file Polyhedron.inlines.hh.

References poly_hull_assign().

Referenced by BFT00_poly_hull_assign_if_exact(), and BHZ09_poly_hull_assign_if_exact().

11.160.5.184 void Parma_Polyhedra_Library::Polyhedron::widening_assign (const Polyhedron & *y*, unsigned * *tp* = 0) [inline]

Same as H79_widening_assign(y, tp).

Definition at line 91 of file Polyhedron.inlines.hh.

References H79_widening_assign().

11.160.5.185 void Parma_Polyhedra_Library::Polyhedron::wrap_assign (const Variables_Set & *vars*, Bounded_Integer_Type_Width *w*, Bounded_Integer_Type_Representation *r*, Bounded_Integer_Type_Overflow *o*, const Constraint_System * *pcs* = 0, unsigned *complexity_threshold* = 16, bool *wrap_individually* = true)

Wraps the specified dimensions of the vector space.

Parameters

vars The set of Variable objects corresponding to the space dimensions to be wrapped.

w The width of the bounded integer type corresponding to all the dimensions to be wrapped.

r The representation of the bounded integer type corresponding to all the dimensions to be wrapped.

o The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.

pcs Possibly null pointer to a constraint system whose variables are contained in `vars`. If `*pcs` depends on variables not in `vars`, the behavior is undefined. When non-null, the pointed-to constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in `*pcs`.

complexity_threshold A precision parameter of the wrapping operator: higher values result in possibly improved precision.

wrap_individually `true` if the dimensions should be wrapped individually (something that results in much greater efficiency to the detriment of precision).

Exceptions

std::invalid_argument Thrown if `*pcs` is dimension-incompatible with `vars`, or if `*this` is dimension-incompatible `vars` or with `*pcs`.

Definition at line 3756 of file Polyhedron_public.cc.

References is_necessarily_closed().

11.160.6 Friends And Related Function Documentation

11.160.6.1 bool operator!= (const Polyhedron & *x*, const Polyhedron & *y*) [related]

Returns `true` if and only if `x` and `y` are different polyhedra.

Note that `x` and `y` may be topology- and/or dimension-incompatible polyhedra: in those cases, the value `true` is returned.

Definition at line 378 of file Polyhedron.inlines.hh.

11.160.6.2 std::ostream & operator<< (std::ostream & *s*, const Polyhedron & *ph*) [related]

Output operator.

Writes a textual representation of `ph` on `s`: `false` is written if `ph` is an empty polyhedron; `true` is written if `ph` is a universe polyhedron; a minimized system of constraints defining `ph` is written otherwise, all constraints in one row separated by ", ".

Definition at line 3777 of file Polyhedron_public.cc.

References is_empty(), and minimized_constraints().

11.160.6.3 bool operator== (const Polyhedron & *x*, const Polyhedron & *y*) [friend]

11.160.6.4 friend class Parma_Polyhedra_Library::BD_Shape [friend]

Definition at line 2495 of file Polyhedron.defs.hh.

11.160.6.5 friend class Parma_Polyhedra_Library::BHRZ03_Certificate [friend]

Definition at line 2498 of file Polyhedron.defs.hh.

11.160.6.6 friend class Parma_Polyhedra_Library::Box [friend]

Definition at line 2494 of file Polyhedron.defs.hh.

11.160.6.7 friend class Parma_Polyhedra_Library::Grid [friend]

Definition at line 2497 of file Polyhedron.defs.hh.

11.160.6.8 friend class Parma_Polyhedra_Library::H79_Certificate [friend]

Definition at line 2499 of file Polyhedron.defs.hh.

11.160.6.9 bool Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces (const Polyhedron &) [friend]

11.160.6.10 friend class Parma_Polyhedra_Library::Octagonal_Shape [friend]

Definition at line 2496 of file Polyhedron.defs.hh.

11.160.6.11 template<typename PH > bool poly_hull_assign_if_exact (PH & *p*, const PH & *q*) [related]

If the poly-hull of `p` and `q` is exact it is assigned to `p` and `true` is returned, otherwise `false` is returned.

Definition at line 52 of file algorithms.hh.

References Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::begin(), contains(), and Parma_Polyhedra_Library::Powerset< Parma_Polyhedra_Library::Determinate< PSET > >::end().

11.160.6.12 void swap (Parma_Polyhedra_Library::Polyhedron & *x*, Parma_Polyhedra_Library::Polyhedron & *y*) [related]

Specializes `std::swap`.

Definition at line 416 of file Polyhedron.inlines.hh.

References swap().

11.160.7 Member Data Documentation

11.160.7.1 Constraint_System Parma_Polyhedra_Library::Polyhedron::con_sys [private]

The system of constraints.

Definition at line 1917 of file Polyhedron.defs.hh.

Referenced by add_recycled_constraints(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), ascii_dump(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_constraints(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), bounded_affine_preimage(), concatenate_assign(), constrains(), constraints(), drop_some_non_integer_points(), external_memory_in_bytes(), H79_widening_assign(), intersection_assign(), is_included_in(), is_necessarily_closed(), is_topologically_closed(), is_universe(), map_space_dimensions(), obtain_sorted_constraints(), obtain_sorted_constraints_with_sat_c(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), OK(), operator=(), Polyhedron(), process_pending_constraints(), process_pending_generators(), quick_equivalence_test(), refine_no_check(), refine_with_constraints(),

relation_with(), remove_higher_space_dimensions(), remove_pending_to_obtain_constraints(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), set_empty(), set_zero_dim_univ(), simplified_constraints(), simplify_using_context_assign(), strongly_minimize_constraints(), strongly_minimize_generators(), swap(), topological_closure_assign(), topology(), update_constraints(), update_generators(), update_sat_c(), and update_sat_g().

11.160.7.2 Generator_System Parma_Polyhedra_Library::Polyhedron::gen_sys [private]

The system of generators.

Definition at line 1920 of file Polyhedron.defs.hh.

Referenced by add_generator(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_and_project(), affine_image(), affine_preimage(), ascii_dump(), BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), bounded_affine_preimage(), bounds(), concatenate_assign(), constrains(), contains_integer_point(), external_memory_in_bytes(), frequency(), generalized_affine_image(), generators(), is_bounded(), is_topologically_closed(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_space_dimensions(), max_min(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), OK(), operator=(), poly_hull_assign(), Polyhedron(), process_pending_constraints(), process_pending_generators(), quick_equivalence_test(), relation_with(), remove_higher_space_dimensions(), remove_pending_to_obtain_generators(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), set_empty(), set_zero_dim_univ(), simplify_using_context_assign(), strongly_minimize_constraints(), strongly_minimize_generators(), swap(), time_elapse_assign(), topological_closure_assign(), unconstrain(), update_constraints(), update_generators(), update_sat_c(), and update_sat_g().

11.160.7.3 Bit_Matrix Parma_Polyhedra_Library::Polyhedron::sat_c [private]

The saturation matrix having constraints on its columns.

Definition at line 1923 of file Polyhedron.defs.hh.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), ascii_dump(), concatenate_assign(), external_memory_in_bytes(), obtain_sorted_constraints(), obtain_sorted_constraints_with_sat_c(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), OK(), operator=(), Polyhedron(), process_pending_constraints(), process_pending_generators(), set_empty(), strongly_minimize_constraints(), strongly_minimize_generators(), swap(), update_constraints(), and update_sat_c().

11.160.7.4 Bit_Matrix Parma_Polyhedra_Library::Polyhedron::sat_g [private]

The saturation matrix having generators on its columns.

Definition at line 1926 of file Polyhedron.defs.hh.

Referenced by add_space_dimensions_and_embed(), add_space_dimensions_and_project(), ascii_dump(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), concatenate_assign(), external_memory_in_bytes(), obtain_sorted_constraints(), obtain_sorted_-

constraints_with_sat_c(), obtain_sorted_generators(), obtain_sorted_generators_with_sat_g(), OK(), operator=(), Polyhedron(), process_pending_constraints(), process_pending_generators(), select_H79_-constraints(), set_empty(), strongly_minimize_constraints(), strongly_minimize_generators(), swap(), update_generators(), and update_sat_g().

11.160.7.5 PPL::dimension_type * Parma_Polyhedra_Library::Polyhedron::simplify_num_-saturators_p = 0 [static, private]

Pointer to an array used by simplify().

Holds (between class initialization and finalization) a pointer to an array, allocated with operator new[ ](), of simplify_num_saturators_size elements.

Definition at line 2484 of file Polyhedron.defs.hh.

Referenced by finalize(), initialize(), and simplify().

11.160.7.6 size_t Parma_Polyhedra_Library::Polyhedron::simplify_num_saturators_size = 0 [static, private]

Dimension of an array used by simplify().

Holds (between class initialization and finalization) the size of the array pointed to by simplify_num_-saturators_p.

Definition at line 2492 of file Polyhedron.defs.hh.

Referenced by finalize(), initialize(), and simplify().

11.160.7.7 dimension_type Parma_Polyhedra_Library::Polyhedron::space_dim [private]

The number of dimensions of the enclosing vector space.

Definition at line 1936 of file Polyhedron.defs.hh.

Referenced by add_congruence(), add_congruences(), add_constraint(), add_generator(), add_recycled_-constraints(), add_recycled_generators(), add_space_dimensions_and_embed(), add_space_dimensions_-and_project(), affine_dimension(), BFT00_poly_hull_assign_if_exact(), BHRZ03_combining_-constraints(), BHRZ03_evolving_points(), BHRZ03_evolving_rays(), BHRZ03_widening_assign(), BHZ09_C_poly_hull_assign_if_exact(), BHZ09_NNC_poly_hull_assign_if_exact(), BHZ09_poly_hull_-assign_if_exact(), bounds(), Parma_Polyhedra_Library::BHRZ03_Certificate::compare(), concatenate_-assign(), constrains(), constraints(), contains(), contains_integer_point(), drop_some_non_integer_-points(), expand_space_dimension(), fold_space_dimensions(), generators(), grid_generators(), H79_-widening_assign(), intersection_assign(), is_bounded(), is_included_in(), is_topologically_closed(), is_universe(), limited_BHRZ03_extrapolation_assign(), limited_H79_extrapolation_assign(), map_-space_dimensions(), max_min(), minimize(), OK(), operator=(), poly_difference_assign(), poly_hull_-assign(), Polyhedron(), process_pending(), process_pending_constraints(), process_pending_generators(), quick_equivalence_test(), refine_no_check(), refine_with_congruence(), refine_with_congruences(), refine_with_constraint(), refine_with_constraints(), relation_with(), remove_higher_space_dimensions(), remove_space_dimensions(), select_CH78_constraints(), select_H79_constraints(), set_zero_dim_univ(), simplify_using_context_assign(), space_dimension(), strongly_minimize_constraints(), strongly_-

minimize_generators(), swap(), time_elapse_assign(), unconstrain(), update_constraints(), and update_-generators().

11.160.7.8 Status Parma_Polyhedra_Library::Polyhedron::status [private]

The status flags to keep track of the polyhedron's internal state.

Definition at line 1933 of file Polyhedron.defs.hh.

Referenced by add_recycled_constraints(), add_space_dimensions_and_embed(), add_space_-dimensions_and_project(), ascii_dump(), clear_constraints_minimized(), clear_constraints_up_-to_date(), clear_empty(), clear_generators_minimized(), clear_generators_up_to_date(), clear_-pending_constraints(), clear_pending_generators(), clear_sat_c_up_to_date(), clear_sat_g_up_to_date(), constraints_are_minimized(), constraints_are_up_to_date(), generators_are_minimized(), generators_-are_up_to_date(), has_pending_constraints(), has_pending_generators(), has_something_pending(), marked_empty(), OK(), operator=(), Polyhedron(), refine_with_constraints(), sat_c_is_up_to_date(), sat_g_is_up_to_date(), set_constraints_minimized(), set_constraints_pending(), set_constraints_up_to_-date(), set_empty(), set_generators_minimized(), set_generators_pending(), set_generators_up_to_date(), set_sat_c_up_to_date(), set_sat_g_up_to_date(), set_zero_dim_univ(), simplify_using_context_assign(), strongly_minimize_constraints(), and swap().

The documentation for this class was generated from the following files:

- Polyhedron.defs.hh
- Polyhedron.inlines.hh
- Polyhedron.templates.hh
- Polyhedron_public.cc
- Polyhedron_nonpublic.cc
- Polyhedron_widenings.cc
- Polyhedron_chdims.cc
- conversion.cc
- minimize.cc
- simplify.cc
- algorithms.hh

11.161 Parma_Polyhedra_Library::Powerset< D > Class Template Reference

The powerset construction on a base-level domain.

`#include <Powerset.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Powerset< D >:

Public Types

- typedef Sequence::size_type size_type
- typedef Sequence::value_type value_type
- typedef iterator_to_const< Sequence > iterator

Alias for a read-only bidirectional iterator on the disjuncts of a Powerset element.

- typedef const_iterator_to_const< Sequence > const_iterator

  A bidirectional const_iterator on the disjuncts of a Powerset element.

- typedef std::reverse_iterator< iterator > reverse_iterator

  The reverse iterator type built from Powerset::iterator.

- typedef std::reverse_iterator< const_iterator > const_reverse_iterator

  The reverse iterator type built from Powerset::const_iterator.

Public Member Functions

Constructors and Destructor

- Powerset ()

  Default constructor: builds the bottom of the powerset constraint system (i.e., the empty powerset).

- Powerset (const Powerset &y)

  Copy constructor.

- Powerset (const D &d)

  If `d` is not bottom, builds a powerset containing only `d`. Builds the empty powerset otherwise.

- ~Powerset ()

  Destructor.

Member Functions that Do Not Modify the Powerset Object

- bool definitely_entails (const Powerset &y) const

  Returns `true` if `*this` definitely entails `y`. Returns `false` if `*this` may not entail `y` (i.e., if `*this` does not entail `y` or if entailment could not be decided).

- bool is_top () const

  Returns `true` if and only if `*this` is the top element of the powerset constraint system (i.e., it represents the universe).

- bool is_bottom () const

  Returns `true` if and only if `*this` is the bottom element of the powerset constraint system (i.e., it represents the empty set).

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns a lower bound to the size in bytes of the memory managed by `*this`.

- bool OK (bool disallow_bottom=false) const

  Checks if all the invariants are satisfied.

Member Functions for the Direct Manipulation of Disjuncts

- void omega_reduce () const

  Drops from the sequence of disjuncts in `*this` *all the non-maximal elements so that* `*this` *is non-redundant.*

- size_type size () const

  Returns the number of disjuncts.

- bool empty () const

  Returns `true` *if and only if there are no disjuncts in* `*this`.

- iterator begin ()

  Returns an iterator pointing to the first disjunct, if `*this` *is not empty; otherwise, returns the past-the-end iterator.*

- iterator end ()

  Returns the past-the-end iterator.

- const_iterator begin () const

  Returns a const_iterator pointing to the first disjunct, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

- reverse_iterator rbegin ()

  Returns a reverse_iterator pointing to the last disjunct, if `*this` *is not empty; otherwise, returns the before-the-start reverse_iterator.*

- reverse_iterator rend ()

  Returns the before-the-start reverse_iterator.

- const_reverse_iterator rbegin () const

  Returns a const_reverse_iterator pointing to the last disjunct, if `*this` *is not empty; otherwise, returns the before-the-start const_reverse_iterator.*

- const_reverse_iterator rend () const

  Returns the before-the-start const_reverse_iterator.

- void add_disjunct (const D &d)

  Adds to `*this` *the disjunct* `d`.

- iterator drop_disjunct (iterator position)

  Drops the disjunct in `*this` *pointed to by* `position`*, returning an iterator to the disjunct following* `position`.

- void drop_disjuncts (iterator first, iterator last)

  Drops all the disjuncts from `first` *to* `last` *(excluded).*

- void clear ()

  Drops all the disjuncts, making `*this` *an empty powerset.*

Member Functions that May Modify the Powerset Object

- Powerset & operator= (const Powerset &y)

The assignment operator.

- void swap (Powerset &y)
  Swaps `*this` *with* `y`.
- void least_upper_bound_assign (const Powerset &y)
  Assigns to `*this` *the least upper bound of* `*this` *and* `y`.
- void upper_bound_assign (const Powerset &y)
  Assigns to `*this` *an upper bound of* `*this` *and* `y`.
- bool upper_bound_assign_if_exact (const Powerset &y)
  Assigns to `*this` *the least upper bound of* `*this` *and* `y` *and returns* `true`.
- void meet_assign (const Powerset &y)
  Assigns to `*this` *the meet of* `*this` *and* `y`.
- void collapse ()
  If `*this` *is not empty (i.e., it is not the bottom element), it is reduced to a singleton obtained by computing an upper-bound of all the disjuncts.*

Protected Types

- typedef std::list< D > Sequence
  A powerset is implemented as a sequence of elements.
- typedef Sequence::iterator Sequence_iterator
  Alias for the low-level iterator on the disjuncts.
- typedef Sequence::const_iterator Sequence_const_iterator
  Alias for the low-level const_iterator on the disjuncts.

Protected Member Functions

- bool is_omega_reduced () const
  Returns `true` *if and only if* `*this` *does not contain non-maximal elements.*
- void collapse (unsigned max_disjuncts)
  Upon return, `*this` *will contain at most* `max_disjuncts` *elements; the set of disjuncts in positions greater than or equal to* `max_disjuncts`*, will be replaced at that position by their upper-bound.*
- iterator add_non_bottom_disjunct_preserve_reduction (const D &d, iterator first, iterator last)
  Adds to `*this` *the disjunct* `d`*, assuming* `d` *is not the bottom element and ensuring partial Omega-reduction.*
- void add_non_bottom_disjunct_preserve_reduction (const D &d)
  Adds to `*this` *the disjunct* `d`*, assuming* `d` *is not the bottom element and preserving Omega-reduction.*
- template<typename Binary_Operator_Assign >
  void pairwise_apply_assign (const Powerset &y, Binary_Operator_Assign op_assign)
  Assigns to `*this` *the result of applying* `op_assign` *pairwise to the elements in* `*this` *and* `y`.

Protected Attributes

- Sequence sequence

  The sequence container holding powerset's elements.

- bool reduced

  If `true`*,* `*this` *is Omega-reduced.*

Private Member Functions

- bool check_omega_reduced () const

  Does the hard work of checking whether `*this` *contains non-maximal elements and returns* `true` *if and only if it does not.*

- void collapse (Sequence_iterator sink)

  Replaces the disjunct `*sink` *by an upper bound of itself and all the disjuncts following it.*

Related Functions

(Note that these are not member functions.)

- template<typename D >
  bool operator== (const Powerset< D > &x, const Powerset< D > &y)

  Returns `true` *if and only if* `x` *and* `y` *are equivalent.*

- template<typename D >
  bool operator!= (const Powerset< D > &x, const Powerset< D > &y)

  Returns `true` *if and only if* `x` *and* `y` *are not equivalent.*

- template<typename D >
  std::ostream & operator<< (std::ostream &s, const Powerset< D > &x)

  Output operator.

- template<typename D >
  void swap (Parma_Polyhedra_Library::Powerset< D > &x, Parma_Polyhedra_Library::Powerset< D > &y)

  Specializes `std::swap`*.*

11.161.1 Detailed Description

template<typename D> class Parma_Polyhedra_Library::Powerset< D >

The powerset construction on a base-level domain. This class offers a generic implementation of a *powerset* domain as defined in Section The Powerset Construction.

Besides invoking the available methods on the disjuncts of a Powerset, this class also provides bidirectional iterators that allow for a direct inspection of these disjuncts. For a consistent handling of Omega-reduction, all the iterators are *read-only*, meaning that the disjuncts cannot be overwritten. Rather, by using the class

iterator, it is possible to drop one or more disjuncts (possibly so as to later add back modified versions). As an example of iterator usage, the following template function drops from powerset ps all the disjuncts that would have become redundant by the addition of an external element d.

```
template <typename D>
void
drop_subsumed(Powerset<D>& ps, const D& d) {
  for (typename Powerset<D>::iterator i = ps.begin(),
         ps_end = ps.end(), i != ps_end; )
    if (i->definitely_entails(d))
      i = ps.drop_disjunct(i);
    else
      ++i;
}
```

The template class D must provide the following methods.

```
memory_size_type total_memory_in_bytes() const
```

Returns a lower bound on the total size in bytes of the memory occupied by the instance of D.

```
bool is_top() const
```

Returns true if and only if the instance of D is the top element of the domain.

```
bool is_bottom() const
```

Returns true if and only if the instance of D is the bottom element of the domain.

```
bool definitely_entails(const D& y) const
```

Returns true if the instance of D definitely entails y. Returns false if the instance may not entail y (i.e., if the instance does not entail y or if entailment could not be decided).

```
void upper_bound_assign(const D& y)
```

Assigns to the instance of D an upper bound of the instance and y.

```
void meet_assign(const D& y)
```

Assigns to the instance of D the meet of the instance and y.

```
bool OK() const
```

Returns true if the instance of D is in a consistent state, else returns false.

The following operators on the template class D must be defined.

```
operator<<(std::ostream& s, const D& x)
```

Writes a textual representation of the instance of D on s.

```
operator==(const D& x, const D& y)
```

Returns true if and only if x and y are equivalent D's.

```
operator!=(const D& x, const D& y)
```

Returns true if and only if x and y are different D's.

Definition at line 147 of file Powerset.defs.hh.

11.161.2 Member Typedef Documentation

11.161.2.1 template<typename D> typedef const_iterator_to_const<Sequence> Parma_Polyhedra_Library::Powerset< D >::const_iterator

A bidirectional const_iterator on the disjuncts of a Powerset element.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 253 of file Powerset.defs.hh.

11.161.2.2 template<typename D> typedef std::reverse_iterator<const_iterator> Parma_Polyhedra_Library::Powerset< D >::const_reverse_iterator

The reverse iterator type built from Powerset::const_iterator.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 259 of file Powerset.defs.hh.

11.161.2.3 template<typename D> typedef iterator_to_const<Sequence> Parma_Polyhedra_Library::Powerset< D >::iterator

Alias for a *read-only* bidirectional iterator on the disjuncts of a Powerset element.

By using this iterator type, the disjuncts cannot be overwritten, but they can be removed using methods `drop_disjunct(iterator position)` and `drop_disjuncts(iterator first, iterator last)`, while still ensuring a correct handling of Omega-reduction.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 250 of file Powerset.defs.hh.

11.161.2.4 template<typename D> typedef std::reverse_iterator<iterator> Parma_Polyhedra_Library::Powerset< D >::reverse_iterator

The reverse iterator type built from Powerset::iterator.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 256 of file Powerset.defs.hh.

11.161.2.5 template<typename D> typedef std::list<D> Parma_Polyhedra_Library::Powerset< D >::Sequence [protected]

A powerset is implemented as a sequence of elements.

The particular sequence employed must support efficient deletion in any position and efficient back insertion.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 221 of file Powerset.defs.hh.

11.161.2.6 template<typename D> typedef Sequence::const_iterator Parma_Polyhedra_Library::Powerset< D >::Sequence_const_iterator [protected]

Alias for the low-level const_iterator on the disjuncts.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 227 of file Powerset.defs.hh.

11.161.2.7 template<typename D> typedef Sequence::iterator Parma_Polyhedra_Library::Powerset< D >::Sequence_iterator [protected]

Alias for the low-level iterator on the disjuncts.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 224 of file Powerset.defs.hh.

11.161.2.8 template<typename D> typedef Sequence::size_type Parma_Polyhedra_Library::Powerset< D >::size_type

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 237 of file Powerset.defs.hh.

11.161.2.9 template<typename D> typedef Sequence::value_type Parma_Polyhedra_Library::Powerset< D >::value_type

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 238 of file Powerset.defs.hh.

11.161.3 Constructor & Destructor Documentation

11.161.3.1 template<typename D > Parma_Polyhedra_Library::Powerset< D >::Powerset () [inline]

Default constructor: builds the bottom of the powerset constraint system (i.e., the empty powerset).

Definition at line 133 of file Powerset.inlines.hh.

11.161.3.2 template<typename D > Parma_Polyhedra_Library::Powerset< D >::Powerset (const Powerset< D > & *y*) [inline]

Copy constructor.

Definition at line 112 of file Powerset.inlines.hh.

11.161.3.3 template<typename D> Parma_Polyhedra_Library::Powerset< D >::Powerset (const D & *d*) [inline, explicit]

If `d` is not bottom, builds a powerset containing only `d`. Builds the empty powerset otherwise.

Definition at line 139 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.3.4 template<typename D > Parma_Polyhedra_Library::Powerset< D >::~Powerset () [inline]

Destructor.

Definition at line 147 of file Powerset.inlines.hh.

11.161.4 Member Function Documentation

11.161.4.1 template<typename D> void Parma_Polyhedra_Library::Powerset< D >::add_disjunct (const D & *d*) [inline]

Adds to `*this` the disjunct `d`.

Definition at line 159 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.2 template<typename D> void Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction (const D & *d*) [inline, protected]

Adds to `*this` the disjunct `d`, assuming `d` is not the bottom element and preserving Omega-reduction.

If `*this` is reduced before an invocation of this method, it will be reduced upon successful return from the method.

Definition at line 152 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::begin(), and Parma_Polyhedra_Library::Powerset< D >::end().

11.161.4.3 template<typename D> Powerset< D >::iterator Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction (const D & *d*, iterator *first*, iterator *last*) [inline, protected]

Adds to `*this` the disjunct `d`, assuming `d` is not the bottom element and ensuring partial Omega-reduction.

If `d` is not the bottom element and is not Omega-redundant with respect to elements in positions between `first` and `last`, all elements in these positions that would be made Omega-redundant by the addition of `d` are dropped and `d` is added to the reduced sequence. If `*this` is reduced before an invocation of this method, it will be reduced upon successful return from the method.

Definition at line 151 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce().

11.161.4.4 template<typename D > Powerset< D >::const_iterator Parma_Polyhedra_Library::Powerset< D >::begin () const [inline]

Returns a const_iterator pointing to the first disjunct, if `*this` is not empty; otherwise, returns the past-the-end const_iterator.

Definition at line 46 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.5 template<typename D > Powerset< D >::iterator Parma_Polyhedra_Library::Powerset< D >::begin () [inline]

Returns an iterator pointing to the first disjunct, if `*this` is not empty; otherwise, returns the past-the-end iterator.

Definition at line 34 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Powerset< D >::check_omega_reduced(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_-

Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Powerset< D >::definitely_entails(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Powerset< D >::external_memory_in_bytes(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers(), Parma_Polyhedra_Library::Powerset< D >::is_top(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe(), Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign(), Parma_Polyhedra_Library::Powerset< D >::OK(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::operator==(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), and Parma_Polyhedra_Library::Powerset< D >::rend().

11.161.4.6 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::check_omega_reduced () const [inline, private]

Does the hard work of checking whether `*this` contains non-maximal elements and returns `true` if and only if it does not.

Definition at line 124 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), and Parma_Polyhedra_Library::Powerset< D >::end().

Referenced by Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced(), and Parma_Polyhedra_Library::Powerset< D >::OK().

11.161.4.7 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::clear () [inline]

Drops all the disjuncts, making `*this` an empty powerset.

Definition at line 106 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.8 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::collapse (Sequence_iterator *sink*) [inline, private]

Replaces the disjunct `*sink` by an upper bound of itself and all the disjuncts following it.

Definition at line 36 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::drop_disjuncts(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::OK(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.9 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::collapse (unsigned *max_disjuncts*) [inline, protected]

Upon return, `*this` will contain at most `max_disjuncts` elements; the set of disjuncts in positions greater than or equal to `max_disjuncts`, will be replaced at that position by their upper-bound.

Definition at line 105 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base, Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced(), Parma_Polyhedra_Library::Powerset< D >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::size().

11.161.4.10 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::collapse () [inline]

If `*this` is not empty (i.e., it is not the bottom element), it is reduced to a singleton obtained by computing an upper-bound of all the disjuncts.

Definition at line 191 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::empty(), and Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_extrapolation_assign(), Parma_Polyhedra_Library::Powerset< D >::collapse(), and Parma_Polyhedra_Library::Powerset< D >::omega_reduce().

11.161.4.11 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::definitely_entails (const Powerset< D > & *y*) const [inline]

Returns `true` if `*this` definitely entails `y`. Returns `false` if `*this` may not entail `y` (i.e., if `*this` does not entail `y` or if entailment could not be decided).

Definition at line 174 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), and Parma_Polyhedra_Library::Powerset< D >::end().

11.161.4.12 template<typename D > Powerset< D >::iterator Parma_Polyhedra_Library::Powerset< D >::drop_disjunct (iterator *position*) [inline]

Drops the disjunct in `*this` pointed to by `position`, returning an iterator to the disjunct following `position`.

Definition at line 94 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base, and Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::operator==().

11.161.4.13 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::drop_disjuncts (iterator *first*, iterator *last*) [inline]

Drops all the disjuncts from `first` to `last` (excluded).

Definition at line 100 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::iterator_to_const< Container >::base, and Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::collapse().

11.161.4.14 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::empty () const [inline]

Returns `true` if and only if there are no disjuncts in `*this`.

Definition at line 88 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::collapse(), and Parma_Polyhedra_Library::Powerset< D >::is_bottom().

11.161.4.15 template<typename D > Powerset< D >::const_iterator Parma_Polyhedra_Library::Powerset< D >::end () const [inline]

Returns the past-the-end const_iterator.

Definition at line 52 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.16 template<typename D > Powerset< D >::iterator Parma_Polyhedra_Library::Powerset< D >::end () [inline]

Returns the past-the-end iterator.

Definition at line 40 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Powerset< D >::check_omega_reduced(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Powerset< D >::definitely_entails(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_

Library::Powerset< D >::external_memory_in_bytes(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::geometrically_covers(), Parma_Polyhedra_Library::Powerset< D >::is_top(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe(), Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign(), Parma_Polyhedra_Library::Powerset< D >::OK(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::operator==(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), and Parma_Polyhedra_Library::Powerset< D >::rbegin().

11.161.4.17 template<typename D > memory_size_type Parma_Polyhedra_Library::Powerset< D >::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by `*this`.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 272 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), and Parma_Polyhedra_Library::Powerset< D >::end().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::external_memory_in_bytes(), and Parma_Polyhedra_Library::Powerset< D >::total_memory_in_bytes().

11.161.4.18 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::is_bottom () const [inline]

Returns `true` if and only if `*this` is the bottom element of the powerset constraint system (i.e., it represents the empty set).

Definition at line 183 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::empty(), and Parma_Polyhedra_Library::Powerset< D >::omega_reduce().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions().

11.161.4.19 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced () const [inline, protected]

Returns `true` if and only if `*this` does not contain non-maximal elements.

Definition at line 143 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::check_omega_reduced(), and Parma_Polyhedra_Library::Powerset< D >::reduced.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::collect_certificates(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe().

11.161.4.20 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::is_top () const [inline]

Returns `true` if and only if `*this` is the top element of the powerset constraint system (i.e., it represents the universe).

Definition at line 173 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), and Parma_Polyhedra_Library::Powerset< D >::omega_reduce().

11.161.4.21 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign (const Powerset< D > & *y*) [inline]

Assigns to `*this` the least upper bound of `*this` and `y`.

Definition at line 235 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::OK(), and Parma_Polyhedra_Library::Powerset< D >::omega_reduce().

Referenced by Parma_Polyhedra_Library::Powerset< D >::upper_bound_assign(), and Parma_Polyhedra_Library::Powerset< D >::upper_bound_assign_if_exact().

11.161.4.22 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::meet_assign (const Powerset< D > & *y*) [inline]

Assigns to `*this` the meet of `*this` and `y`.

Definition at line 198 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign().

11.161.4.23 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::OK (bool *disallow_bottom* = `false`) const [inline]

Checks if all the invariants are satisfied.

Definition at line 286 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::check_omega_reduced(), Parma_Polyhedra_Library::Powerset< D >::end(), and Parma_Polyhedra_Library::Powerset< D >::reduced.

Referenced by Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), and Parma_Polyhedra_Library::Powerset< D >::Powerset().

11.161.4.24 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::omega_reduce () const [inline]

Drops from the sequence of disjuncts in *this all the non-maximal elements so that *this is non-redundant.

This method is declared const because, even though Omega-reduction may change the syntactic representation of *this, its semantics will be unchanged.

Definition at line 60 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::abandon_expensive_computations, Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::OK(), and Parma_Polyhedra_Library::Powerset< D >::reduced.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_above(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_below(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::concatenate_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Powerset< D >::is_bottom(), Parma_Polyhedra_Library::Powerset< D >::is_top(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_topologically_closed(), Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize(), Parma_Polyhedra_Library::Powerset< D >::operator==(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::strictly_contains().

11.161.4.25 template<typename D > Powerset< D > & Parma_Polyhedra_Library::Powerset< D >::operator= (const Powerset< D > & *y*) [inline]

The assignment operator.

Definition at line 118 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

11.161.4.26 template<typename D > template<typename Binary_Operator_Assign > void Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign (const Powerset< D > & *y*, Binary_Operator_Assign *op_assign*) [inline, protected]

Assigns to *this the result of applying op_assign pairwise to the elements in *this and y.

The elements of the powerset result are obtained by applying op_assign to each pair of elements whose components are drawn from *this and y, respectively.

Definition at line 213 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::reduced, Parma_Polyhedra_Library::Powerset< D >::sequence, and Parma_Polyhedra_Library::Powerset< D >::swap().

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_assign(), Parma_Polyhedra_Library::Powerset< D >::meet_assign(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::time_elapse_assign().

11.161.4.27 template<typename D > Powerset< D >::const_reverse_iterator Parma_Polyhedra_Library::Powerset< D >::rbegin () const [inline]

Returns a const_reverse_iterator pointing to the last disjunct, if `*this` is not empty; otherwise, returns the before-the-start const_reverse_iterator.

Definition at line 70 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::end().

11.161.4.28 template<typename D > Powerset< D >::reverse_iterator Parma_Polyhedra_Library::Powerset< D >::rbegin () [inline]

Returns a reverse_iterator pointing to the last disjunct, if `*this` is not empty; otherwise, returns the before-the-start reverse_iterator.

Definition at line 58 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::end().

11.161.4.29 template<typename D > Powerset< D >::const_reverse_iterator Parma_Polyhedra_Library::Powerset< D >::rend () const [inline]

Returns the before-the-start const_reverse_iterator.

Definition at line 76 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin().

11.161.4.30 template<typename D > Powerset< D >::reverse_iterator Parma_Polyhedra_Library::Powerset< D >::rend () [inline]

Returns the before-the-start reverse_iterator.

Definition at line 64 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin().

11.161.4.31 template<typename D > Powerset< D >::size_type Parma_Polyhedra_Library::Powerset< D >::size () const [inline]

Returns the number of disjuncts.

Definition at line 82 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::ascii_dump(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::BHZ03_widening_assign(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_universe(), Parma_Polyhedra_Library::Powerset< D >::operator==(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign().

11.161.4.32 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::swap (Powerset< D > & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 126 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::reduced, and Parma_Polyhedra_Library::Powerset< D >::sequence.

Referenced by Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), and Parma_Polyhedra_Library::Powerset< D >::swap().

11.161.4.33 template<typename D > memory_size_type Parma_Polyhedra_Library::Powerset< D >::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented in Parma_Polyhedra_Library::Pointset_Powerset< PSET >.

Definition at line 217 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::external_memory_in_bytes().

11.161.4.34 template<typename D > void Parma_Polyhedra_Library::Powerset< D >::upper_bound_assign (const Powerset< D > & *y*) [inline]

Assigns to `*this` an upper bound of `*this` and `y`.

The result will be the least upper bound of `*this` and `y`.

Definition at line 204 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign().

11.161.4.35 template<typename D > bool Parma_Polyhedra_Library::Powerset< D >::upper_bound_assign_if_exact (const Powerset< D > & *y*) [inline]

Assigns to `*this` the least upper bound of `*this` and `y` and returns `true`.

Exceptions

std::invalid_argument Thrown if `*this` and `y` are dimension-incompatible.

Definition at line 210 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::least_upper_bound_assign().

11.161.5 Friends And Related Function Documentation

11.161.5.1 template<typename D > bool operator!= (const Powerset< D > & *x*, const Powerset< D > & *y*) [related]

Returns `true` if and only if `x` and `y` are not equivalent.

Definition at line 167 of file Powerset.inlines.hh.

11.161.5.2 template<typename D > std::ostream & operator<< (std::ostream & *s*, const Powerset< D > & *x*) [related]

Output operator.

Definition at line 253 of file Powerset.templates.hh.

11.161.5.3 template<typename D > bool operator== (const Powerset< D > & *x*, const Powerset< D > & *y*) [related]

Returns `true` if and only if `x` and `y` are equivalent.

Definition at line 190 of file Powerset.templates.hh.

References Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), and Parma_Polyhedra_Library::Powerset< D >::size().

11.161.5.4 template<typename D > void swap (Parma_Polyhedra_Library::Powerset< D > & *x*, Parma_Polyhedra_Library::Powerset< D > & *y*) [related]

Specializes `std::swap`.

Definition at line 229 of file Powerset.inlines.hh.

References Parma_Polyhedra_Library::Powerset< D >::swap().

11.161.6 Member Data Documentation

11.161.6.1 template<typename D> bool Parma_Polyhedra_Library::Powerset< D >::reduced [mutable, protected]

If `true`, `*this` is Omega-reduced.

Definition at line 233 of file Powerset.defs.hh.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraints(), Parma_Polyhedra_Library::Powerset< D >::add_disjunct(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_preimage(), Parma_Polyhedra_Library::Powerset< D >::is_omega_reduced(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions(), Parma_Polyhedra_Library::Powerset< D >::OK(), Parma_Polyhedra_Library::Powerset< D >::omega_reduce(), Parma_Polyhedra_Library::Powerset< D >::operator=(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraints(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign(), Parma_Polyhedra_Library::Powerset< D >::swap(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::unconstrain(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::wrap_assign().

11.161.6.2 template<typename D> Sequence Parma_Polyhedra_Library::Powerset< D >::sequence [protected]

The sequence container holding powerset's elements.

Definition at line 230 of file Powerset.defs.hh.

Referenced by Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_constraints(), Parma_Polyhedra_Library::Powerset< D >::add_disjunct(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_disjunct(), Parma_Polyhedra_Library::Powerset< D >::add_non_bottom_disjunct_preserve_reduction(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::add_space_dimensions_and_project(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::affine_preimage(), Parma_Polyhedra_Library::Powerset< D >::begin(), Parma_Polyhedra_Library::Pointset_-

Powerset< PSET >::BGP99_heuristics_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounded_affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_above(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::bounds_from_below(), Parma_Polyhedra_Library::Powerset< D >::clear(), Parma_Polyhedra_Library::Powerset< D >::collapse(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::contains(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::contains_integer_point(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::difference_assign(), Parma_Polyhedra_Library::Powerset< D >::drop_disjunct(), Parma_Polyhedra_Library::Powerset< D >::drop_disjuncts(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Powerset< D >::empty(), Parma_Polyhedra_Library::Powerset< D >::end(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::expand_space_dimension(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::fold_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_image(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::generalized_affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::intersection_preserving_enlarge_element(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_bounded(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_discrete(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_disjoint_from(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_empty(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::is_topologically_closed(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::map_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize(), Parma_Polyhedra_Library::Powerset< D >::operator=(), Parma_Polyhedra_Library::Powerset< D >::pairwise_apply_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::pairwise_reduce(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::Pointset_Powerset(), Parma_Polyhedra_Library::Powerset< D >::Powerset(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruence(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_congruences(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraint(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::refine_with_constraints(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::relation_with(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_higher_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::remove_space_dimensions(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::simplify_using_context_assign(), Parma_Polyhedra_Library::Powerset< D >::size(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::strictly_contains(), Parma_Polyhedra_Library::Powerset< D >::swap(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::topological_closure_assign(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::unconstrain(), and Parma_Polyhedra_Library::Pointset_Powerset< PSET >::wrap_assign().

The documentation for this class was generated from the following files:

- Powerset.defs.hh
- Powerset.inlines.hh
- Powerset.templates.hh

11.162 Parma_Polyhedra_Library::Interval_NS::Property Struct Reference

`#include <Interval_Info.defs.hh>`

Public Types

- enum Type { CARDINALITY_0_, CARDINALITY_1_, CARDINALITY_IS_ }
- typedef bool Value

Public Member Functions

- Property (Type t)

Public Attributes

- Type type

Static Public Attributes

- static const Value default_value = true
- static const Value unsupported_value = false

11.162.1 Detailed Description

Definition at line 36 of file Interval_Info.defs.hh.

11.162.2 Member Typedef Documentation

11.162.2.1 typedef bool Parma_Polyhedra_Library::Interval_NS::Property::Value

Definition at line 42 of file Interval_Info.defs.hh.

11.162.3 Member Enumeration Documentation

11.162.3.1 enum Parma_Polyhedra_Library::Interval_NS::Property::Type

Enumerator:

CARDINALITY_0_

CARDINALITY_1_

CARDINALITY_IS_

Definition at line 37 of file Interval_Info.defs.hh.

11.162.4 Constructor & Destructor Documentation

11.162.4.1 Parma_Polyhedra_Library::Interval_NS::Property::Property (Type *t*) [inline]

Definition at line 45 of file Interval_Info.defs.hh.

11.162.5 Member Data Documentation

11.162.5.1 const Value Parma_Polyhedra_Library::Interval_NS::Property::default_value = true [static]

Definition at line 43 of file Interval_Info.defs.hh.

11.162.5.2 Type Parma_Polyhedra_Library::Interval_NS::Property::type

Definition at line 48 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::get_interval_property(), and Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >::set_interval_property().

11.162.5.3 const Value Parma_Polyhedra_Library::Interval_NS::Property::unsupported_value = false [static]

Definition at line 44 of file Interval_Info.defs.hh.

Referenced by Parma_Polyhedra_Library::Interval_Info_Null< Policy >::get_boundary_property(), and Parma_Polyhedra_Library::Interval_Info_Null< Policy >::get_interval_property().

The documentation for this struct was generated from the following file:

- Interval_Info.defs.hh

11.163 Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U > Class Template Reference

An object that behaves like a matrix's row with respect to the subscript operators.

Public Member Functions

- template<typename V >
  Pseudo_Row (const Pseudo_Row< V > &y)

  Copy constructor allowing the construction of a const pseudo-row from a non-const pseudo-row. Ordinary copy constructor.

- ~Pseudo_Row ()

  Destructor.

- U & operator[ ] (dimension_type k) const

  Subscript operator.

- Pseudo_Row ()

  Default constructor: creates an invalid object that has to be assigned.

- Pseudo_Row & operator= (const Pseudo_Row &y)

  Assignment operator.

Private Member Functions

- Pseudo_Row (U &y)

  Private constructor for a Pseudo_Row beginning at y.

Private Attributes

- U * first

  Holds a reference to the beginning of this row.

Friends

- class Pseudo_Row
- class any_row_iterator
- class OR_Matrix

11.163.1 Detailed Description

template<typename T>template<typename U> class Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >

An object that behaves like a matrix's row with respect to the subscript operators.

Definition at line 113 of file OR_Matrix.defs.hh.

11.163.2 Constructor & Destructor Documentation

11.163.2.1 template<typename T > template<typename U > template<typename V > Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::Pseudo_Row (const Pseudo_Row< V > & *y*) [inline]

Copy constructor allowing the construction of a const pseudo-row from a non-const pseudo-row. Ordinary copy constructor.

Definition at line 91 of file OR_Matrix.inlines.hh.

11.163.2.2 template<typename T > template<typename U > Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::~Pseudo_Row () [inline]

Destructor.

Definition at line 113 of file OR_Matrix.inlines.hh.

11.163.2.3 template<typename T> template<typename U> Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::Pseudo_Row ()

Default constructor: creates an invalid object that has to be assigned.

11.163.2.4 template<typename T > template<typename U > Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::Pseudo_Row (U & *y*) [inline, explicit, private]

Private constructor for a Pseudo_Row beginning at `y`.

Definition at line 75 of file OR_Matrix.inlines.hh.

11.163.3 Member Function Documentation

11.163.3.1 template<typename T > template<typename U > OR_Matrix< T >::Pseudo_Row< U > & Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::operator= (const Pseudo_Row< U > & *y*) [inline]

Assignment operator.

Definition at line 102 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::first.

11.163.3.2 template<typename T > template<typename U > U & Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::operator[ ] (dimension_type *k*) const [inline]

Subscript operator.

Definition at line 119 of file OR_Matrix.inlines.hh.

References Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::first.

11.163.4 Friends And Related Function Documentation

11.163.4.1 template<typename T> template<typename U> friend class any_row_iterator [friend]

Definition at line 183 of file OR_Matrix.defs.hh.

11.163.4.2 template<typename T> template<typename U> friend class OR_Matrix [friend]

Definition at line 186 of file OR_Matrix.defs.hh.

11.163.4.3 template<typename T> template<typename U> Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::Pseudo_Row [friend]

Definition at line 182 of file OR_Matrix.defs.hh.

11.163.5 Member Data Documentation

11.163.5.1 template<typename T> template<typename U> U* Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::first [private]

Holds a reference to the beginning of this row.

Definition at line 155 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::operator=(), and Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >::operator[ ]().

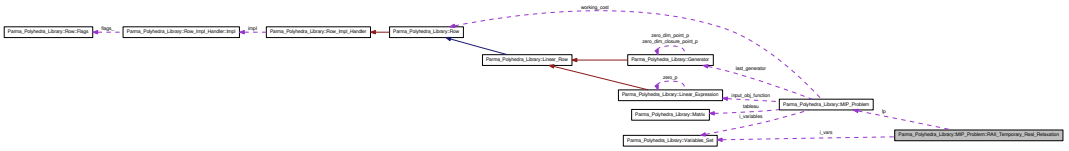
The documentation for this class was generated from the following files:

- OR_Matrix.defs.hh
- OR_Matrix.inlines.hh

11.164 Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation Struct Reference

A helper class to temporarily relax a MIP problem using RAII.

Collaboration diagram for Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation:



Public Member Functions

- RAII_Temporary_Real_Relaxation (MIP_Problem &mip)
- ~RAII_Temporary_Real_Relaxation ()

Public Attributes

- MIP_Problem & lp
- Variables_Set i_vars

11.164.1 Detailed Description

A helper class to temporarily relax a MIP problem using RAII.

Definition at line 510 of file MIP_Problem.defs.hh.

11.164.2 Constructor & Destructor Documentation

11.164.2.1 Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation::RAII_Temporary_Real_Relaxation (MIP_Problem & *mip*) [inline]

Definition at line 514 of file MIP_Problem.defs.hh.

References Parma_Polyhedra_Library::MIP_Problem::i_variables, i_vars, lp, and Parma_Polyhedra_Library::MIP_Problem::swap().

11.164.2.2 Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation::~RAII_Temporary_Real_Relaxation () [inline]

Definition at line 520 of file MIP_Problem.defs.hh.

References Parma_Polyhedra_Library::MIP_Problem::i_variables, i_vars, lp, and Parma_Polyhedra_Library::MIP_Problem::swap().

11.164.3 Member Data Documentation

11.164.3.1 Variables_Set Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation::i_vars

Definition at line 512 of file MIP_Problem.defs.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::is_satisfiable(), RAII_Temporary_Real_Relaxation(), Parma_Polyhedra_Library::MIP_Problem::solve(), and ~RAII_Temporary_Real_Relaxation().

11.164.3.2 MIP_Problem& Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation::lp

Definition at line 511 of file MIP_Problem.defs.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::is_satisfiable(), RAII_Temporary_Real_-Relaxation(), Parma_Polyhedra_Library::MIP_Problem::solve(), and ~RAII_Temporary_Real_-Relaxation().

The documentation for this struct was generated from the following file:

- MIP_Problem.defs.hh

11.165 Parma_Polyhedra_Library::Rectilinear_Distance_Specialization< Temp > Struct Template Reference

`#include <distances.defs.hh>`

Static Public Member Functions

- static void combine (Temp &running, const Temp &current, Rounding_Dir dir)
- static void finalize (Temp &, Rounding_Dir)

11.165.1 Detailed Description

template<typename Temp> struct Parma_Polyhedra_Library::Rectilinear_Distance_-Specialization< Temp >

Definition at line 31 of file distances.defs.hh.

11.165.2 Member Function Documentation

11.165.2.1 template<typename Temp > void Parma_Polyhedra_Library::Rectilinear_Distance_-Specialization< Temp >::combine (Temp & *running*, const Temp & *current*, Rounding_Dir *dir*) [inline, static]

Definition at line 70 of file distances.inlines.hh.

11.165.2.2 template<typename Temp > void Parma_Polyhedra_Library::Rectilinear_-Distance_Specialization< Temp >::finalize (Temp &, Rounding_Dir) [inline, static]

Definition at line 78 of file distances.inlines.hh.

The documentation for this struct was generated from the following files:

- distances.defs.hh
- distances.inlines.hh

11.166 Parma_Polyhedra_Library::Recycle_Input Struct Reference

A tag class.

```
#include <globals.defs.hh>
```

11.166.1 Detailed Description

A tag class. Tag class to distinguish those constructors that recycle the data structures of their arguments, instead of taking a copy.

Definition at line 233 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

- globals.defs.hh

11.167 Parma_Polyhedra_Library::Determinate< PSET >::Rep Class Reference

The possibly shared representation of a Determinate object.

Public Member Functions

- Rep (dimension_type num_dimensions, Degenerate_Element kind)

  Builds a new representation by creating a pointset of the specified kind, in the specified vector space.

- Rep (const PSET &p)

  Builds a new representation by copying the pointset `p`.

- Rep (const Constraint_System &cs)

  Builds a new representation by copying the constraints in `cs`.

- Rep (const Congruence_System &cgs)

  Builds a new representation by copying the constraints in `cgs`.

- ~Rep ()

  Destructor.

- void new_reference () const

  Registers a new reference.

- bool del_reference () const

  Unregisters one reference; returns `true` *if and only if the representation has become unreferenced.*

- bool is_shared () const

  True if and only if this representation is currently shared.

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns a lower bound to the size in bytes of the memory managed by `*this`.

Public Attributes

- PSET pset
  The possibly shared, embedded pointset.

Private Member Functions

- Rep & operator= (const Rep &y)
  Private and unimplemented: assignment not allowed.
- Rep (const Rep &y)
  Private and unimplemented: copies not allowed.
- Rep ()
  Private and unimplemented: default construction not allowed.

Private Attributes

- unsigned long references
  Count the number of references:
  - *0: leaked,* `pset` *is non-const;*
  - *1: one reference,* `pset` *is non-const;*
  - *> 1: more than one reference,* `pset` *is const.*

11.167.1 Detailed Description

template<typename PSET> class Parma_Polyhedra_Library::Determinate< PSET >::Rep

The possibly shared representation of a Determinate object. By adopting the *copy-on-write* technique, a single representation of the base-level object may be shared by more than one object of the class Determinate.

Definition at line 245 of file Determinate.defs.hh.

11.167.2 Constructor & Destructor Documentation

11.167.2.1 template<typename PSET> Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep (const Rep & *y*) [private]

Private and unimplemented: copies not allowed.

11.167.2.2 template<typename PSET> Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep () [private]

Private and unimplemented: default construction not allowed.

11.167.2.3 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep (dimension_type *num_dimensions*, Degenerate_Element *kind*) [inline]

Builds a new representation by creating a pointset of the specified kind, in the specified vector space.

Definition at line 33 of file Determinate.inlines.hh.

11.167.2.4 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep (const PSET & *p*) [inline]

Builds a new representation by copying the pointset `p`.

Definition at line 40 of file Determinate.inlines.hh.

11.167.2.5 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep (const Constraint_System & *cs*) [inline]

Builds a new representation by copying the constraints in `cs`.

Definition at line 46 of file Determinate.inlines.hh.

11.167.2.6 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Rep::Rep (const Congruence_System & *cgs*) [inline]

Builds a new representation by copying the constraints in `cgs`.

Definition at line 52 of file Determinate.inlines.hh.

11.167.2.7 template<typename PSET > Parma_Polyhedra_Library::Determinate< PSET >::Rep::~Rep () [inline]

Destructor.

Definition at line 58 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::references.

11.167.3 Member Function Documentation

11.167.3.1 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::Rep::del_reference () const [inline]

Unregisters one reference; returns `true` if and only if the representation has become unreferenced.

Definition at line 70 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::references.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::mutate(), Parma_Polyhedra_Library::Determinate< PSET >::operator=(), and Parma_Polyhedra_Library::Determinate< PSET >::~Determinate().

11.167.3.2 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Determinate< PSET >::Rep::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by `*this`.

Definition at line 82 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::Rep::total_memory_in_bytes().

11.167.3.3 template<typename PSET > bool Parma_Polyhedra_Library::Determinate< PSET >::Rep::is_shared () const [inline]

True if and only if this representation is currently shared.

Definition at line 76 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::references.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::mutate().

11.167.3.4 template<typename PSET > void Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference () const [inline]

Registers a new reference.

Definition at line 64 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::references.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::Determinate(), Parma_Polyhedra_Library::Determinate< PSET >::mutate(), and Parma_Polyhedra_Library::Determinate< PSET >::operator=().

11.167.3.5 template<typename PSET> Rep& Parma_Polyhedra_Library::Determinate< PSET >::Rep::operator= (const Rep & *y*) [private]

Private and unimplemented: assignment not allowed.

11.167.3.6 template<typename PSET > memory_size_type Parma_Polyhedra_Library::Determinate< PSET >::Rep::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Definition at line 88 of file Determinate.inlines.hh.

References Parma_Polyhedra_Library::Determinate< PSET >::Rep::external_memory_in_bytes().

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::external_memory_in_bytes().

11.167.4 Member Data Documentation

11.167.4.1 template<typename PSET> PSET Parma_Polyhedra_Library::Determinate< PSET >::Rep::pset

The possibly shared, embedded pointset.

Definition at line 266 of file Determinate.defs.hh.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::definitely_entails(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::external_memory_in_bytes(), Parma_Polyhedra_Library::Determinate< PSET >::is_bottom(), Parma_Polyhedra_Library::Determinate< PSET >::is_definitely_equivalent_to(), Parma_Polyhedra_Library::Determinate< PSET >::is_top(), Parma_Polyhedra_Library::Determinate< PSET >::mutate(), Parma_Polyhedra_Library::Determinate< PSET >::OK(), and Parma_Polyhedra_Library::Determinate< PSET >::pointset().

11.167.4.2 template<typename PSET> unsigned long Parma_Polyhedra_Library::Determinate< PSET >::Rep::references [mutable, private]

Count the number of references:

- 0: leaked, `pset` is non-const;
- 1: one reference, `pset` is non-const;
- > 1: more than one reference, `pset` is const.

Definition at line 253 of file Determinate.defs.hh.

Referenced by Parma_Polyhedra_Library::Determinate< PSET >::Rep::del_reference(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::is_shared(), Parma_Polyhedra_Library::Determinate< PSET >::Rep::new_reference(), and Parma_Polyhedra_Library::Determinate< PSET >::Rep::~Rep().

The documentation for this class was generated from the following files:

- Determinate.defs.hh
- Determinate.inlines.hh

11.168 Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer type

Static Public Member Functions

- static type get (const From &x)

11.168.1 Detailed Description

template<typename From, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >

Definition at line 204 of file Interval_Restriction.defs.hh.

11.168.2 Member Typedef Documentation

11.168.2.1 template<typename From , typename Base > typedef Simple_Restriction_Integer Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::type

Definition at line 205 of file Interval_Restriction.defs.hh.

11.168.3 Member Function Documentation

11.168.3.1 template<typename From , typename Base > static type Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 206 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::is_integer().

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.169 Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Interval_Restriction_Integer< Base > type

Static Public Member Functions

- static const type & get (const From &x)

11.169.1 Detailed Description

template<typename From, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >

Definition at line 220 of file Interval_Restriction.defs.hh.

11.169.2 Member Typedef Documentation

11.169.2.1 template<typename From , typename Base > typedef Interval_Restriction_Integer<Base> Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >::type

Definition at line 221 of file Interval_Restriction.defs.hh.

11.169.3 Member Function Documentation

11.169.3.1 template<typename From , typename Base > static const type& Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 222 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.170 Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer type

Static Public Member Functions

- static type get (const From &x)

11.170.1 Detailed Description

template<typename From, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >

Definition at line 407 of file Interval_Restriction.defs.hh.

11.170.2 Member Typedef Documentation

11.170.2.1 template<typename From , typename Base > typedef Simple_Restriction_Integer Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >::type

Definition at line 408 of file Interval_Restriction.defs.hh.

11.170.3 Member Function Documentation

11.170.3.1 template<typename From , typename Base > static type Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 409 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.171 Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer type

Static Public Member Functions

- static type get (const From &x)

11.171.1 Detailed Description

template<typename From, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >

Definition at line 212 of file Interval_Restriction.defs.hh.

11.171.2 Member Typedef Documentation

11.171.2.1 template<typename From , typename Base > typedef Simple_Restriction_Integer Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::type

Definition at line 213 of file Interval_Restriction.defs.hh.

11.171.3 Member Function Documentation

11.171.3.1 template<typename From , typename Base > static type Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 214 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::is_integer().

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.172 Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer_Modulo< T > type

Static Public Member Functions

- static const type & get (const From &x)

11.172.1 Detailed Description

template<typename From, typename T, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >

Definition at line 429 of file Interval_Restriction.defs.hh.

11.172.2 Member Typedef Documentation

11.172.2.1 template<typename From , typename T , typename Base > typedef Simple_Restriction_Integer_Modulo<T> Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::type

Definition at line 430 of file Interval_Restriction.defs.hh.

11.172.3 Member Function Documentation

11.172.3.1 template<typename From , typename T , typename Base > static const type& Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 431 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::is_integer().

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.173 Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer_Modulo< T > type

Static Public Member Functions

- static const type & get (const From &x)

11.173.1 Detailed Description

template<typename From, typename T, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >

Definition at line 455 of file Interval_Restriction.defs.hh.

11.173.2 Member Typedef Documentation

11.173.2.1 template<typename From , typename T , typename Base > typedef Simple_Restriction_Integer_Modulo<T> Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >::type

Definition at line 456 of file Interval_Restriction.defs.hh.

11.173.3 Member Function Documentation

11.173.3.1 template<typename From , typename T , typename Base > static const type& Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 457 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.174 Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Interval_Restriction_Integer_Modulo< T, Base > type

Static Public Member Functions

- static const type & get (const From &x)

11.174.1 Detailed Description

template<typename From, typename T, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >

Definition at line 468 of file Interval_Restriction.defs.hh.

11.174.2 Member Typedef Documentation

11.174.2.1 template<typename From , typename T , typename Base > typedef Interval_Restriction_Integer_Modulo<T, Base> Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >::type

Definition at line 469 of file Interval_Restriction.defs.hh.

11.174.3 Member Function Documentation

11.174.3.1 template<typename From , typename T , typename Base > static const type& Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 470 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.175 Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Public Types

- typedef Simple_Restriction_Integer_Modulo< T > type

Static Public Member Functions

- static const type & get (const From &x)

11.175.1 Detailed Description

template<typename From, typename T, typename Base> struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >

Definition at line 442 of file Interval_Restriction.defs.hh.

11.175.2 Member Typedef Documentation

11.175.2.1 template<typename From , typename T , typename Base > typedef Simple_Restriction_Integer_Modulo<T> Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::type

Definition at line 443 of file Interval_Restriction.defs.hh.

11.175.3 Member Function Documentation

11.175.3.1 template<typename From , typename T , typename Base > static const type& Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >::get (const From & *x*) `[inline, static]`

Definition at line 444 of file Interval_Restriction.defs.hh.

References Parma_Polyhedra_Library::is_integer().

The documentation for this struct was generated from the following file:

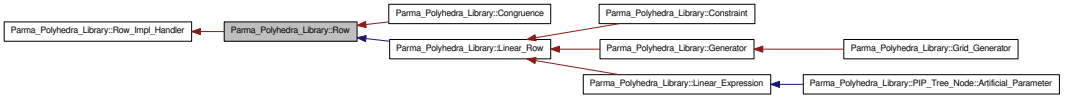
- Interval_Restriction.defs.hh

11.176 Parma_Polyhedra_Library::Row Class Reference

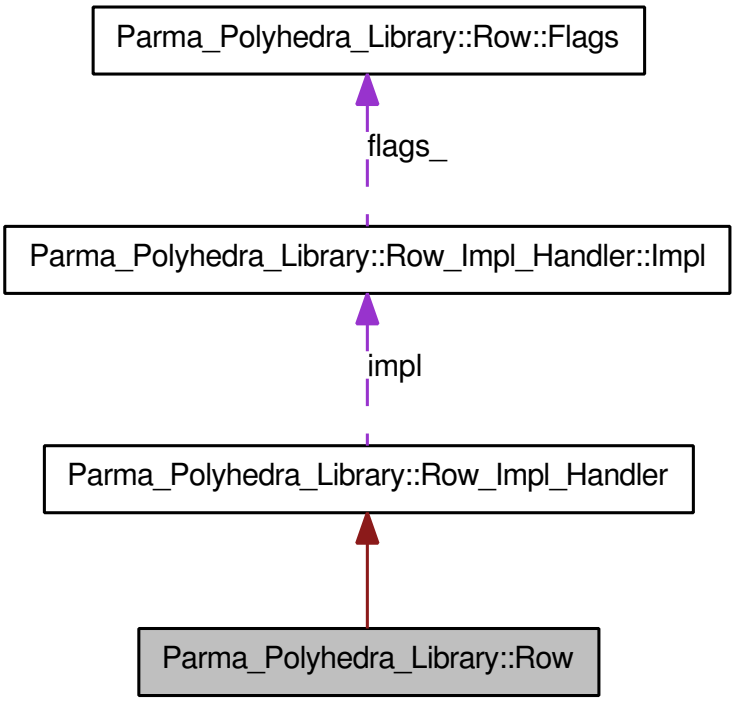
A finite sequence of coefficients.

`#include <Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Row:



Collaboration diagram for Parma_Polyhedra_Library::Row:



Classes

- class Flags

  Wrapper class to represent a set of flags with bits in a native unsigned integral type.

Public Member Functions

- Row ()

  Pre-constructs a row: construction must be completed by construct().

- Row (dimension_type sz, Flags f)

  Tight constructor: resizing may require reallocation.

- Row (dimension_type sz, dimension_type capacity, Flags f)

  Sizing constructor with capacity.

- Row (const Row &y)

  Ordinary copy constructor.

- Row (const Row &y, dimension_type capacity)

  Copy constructor with specified capacity.

- Row (const Row &y, dimension_type sz, dimension_type capacity)

  Copy constructor with specified size and capacity.

- ~Row ()

  Destructor.

- Row & operator= (const Row &y)

  Assignment operator.

- void swap (Row &y)

  Swaps `*this` *with* `y`.

- void assign (Row &y)

  Assigns the implementation of `y` *to* `*this`.

- void allocate (dimension_type capacity, Flags f)

  Allocates memory for a default constructed Row object, setting flags to `f` *and allowing for* `capacity` *coefficients at most.*

- void expand_within_capacity (dimension_type new_size)

  Expands the row to size `new_size`.

- void shrink (dimension_type new_size)

  Shrinks the row by erasing elements at the end.

- const Flags & flags () const

  Returns a const reference to the flags of `*this`.

- Flags & flags ()

  Returns a non-const reference to the flags of `*this`.

- dimension_type size () const

  Gives the number of coefficients currently in use.

- void normalize ()

  Normalizes the modulo of coefficients so that they are mutually prime.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns a lower bound to the size in bytes of the memory managed by `*this`.

- memory_size_type total_memory_in_bytes (dimension_type capacity) const

  Returns the total size in bytes of the memory occupied by `*this`*, provided the capacity of* `*this` *is given by* `capacity`.

- memory_size_type external_memory_in_bytes (dimension_type capacity) const

  Returns the size in bytes of the memory managed by `*this`*, provided the capacity of* `*this` *is given by* `capacity`.

- bool OK () const

  Checks if all the invariants are satisfied.

- bool OK (dimension_type row_size, dimension_type row_capacity) const

  Checks if all the invariants are satisfied and that the actual size and capacity match the values provided as arguments.

Post-constructors

- void construct (dimension_type sz, Flags f)

  Constructs properly a default-constructed element.

- void construct (dimension_type sz, dimension_type capacity, Flags f)

  Constructs properly a default-constructed element.

Subscript operators

- Coefficient & operator[ ] (dimension_type k)

  Returns a reference to the element of the row indexed by `k`.

- Coefficient_traits::const_reference operator[ ] (dimension_type k) const

  Returns a constant reference to the element of the row indexed by `k`.

Static Public Member Functions

- static dimension_type max_size ()

  Returns the size() of the largest possible Row.

Private Member Functions

- void copy_construct_coefficients (const Row &y)

  Exception-safe copy construction mechanism for coefficients.

Related Functions

(Note that these are not member functions.)

- bool operator== (const Row &x, const Row &y)

  Returns `true` *if and only if* `x` *and* `y` *are equal.*

- bool operator!= (const Row &x, const Row &y)

  Returns `true` *if and only if* `x` *and* `y` *are different.*

- void swap (Parma_Polyhedra_Library::Row &x, Parma_Polyhedra_Library::Row &y)

  Specializes `std::swap`.

- void iter_swap (std::vector< Parma_Polyhedra_Library::Row >::iterator x, std::vector< Parma_Polyhedra_Library::Row >::iterator y)

  Specializes `std::iter_swap`.

11.176.1 Detailed Description

A finite sequence of coefficients.

Definition at line 89 of file Row.defs.hh.

11.176.2 Constructor & Destructor Documentation

11.176.2.1 Parma_Polyhedra_Library::Row::Row () [inline]

Pre-constructs a row: construction must be completed by construct().

Definition at line 207 of file Row.inlines.hh.

11.176.2.2 Parma_Polyhedra_Library::Row::Row (dimension_type *sz*, Flags *f*) [inline]

Tight constructor: resizing may require reallocation.

Constructs a row with size and capacity `sz`, and flags `f`.

Definition at line 272 of file Row.inlines.hh.

References construct().

11.176.2.3 Parma_Polyhedra_Library::Row::Row (dimension_type *sz*, dimension_type *capacity*, Flags *f*) [inline]

Sizing constructor with capacity.

Parameters

sz The size of the row that will be constructed;

capacity The capacity of the row that will be constructed;

f Flags for the row that will be constructed.

The row that is constructed has storage for `capacity` elements, `sz` of which are default-constructed now. The row flags are set to `f`.

Definition at line 264 of file Row.inlines.hh.

References construct().

11.176.2.4 Parma_Polyhedra_Library::Row::Row (const Row & *y*) [inline]

Ordinary copy constructor.

Definition at line 278 of file Row.inlines.hh.

References allocate(), Parma_Polyhedra_Library::compute_capacity(), copy_construct_coefficients(), flags(), Parma_Polyhedra_Library::Row_Impl_Handler::impl, max_size(), and size().

11.176.2.5 Parma_Polyhedra_Library::Row::Row (const Row & *y*, dimension_type *capacity*) [inline]

Copy constructor with specified capacity.

It is assumed that `capacity` is greater than or equal to the size of `y`.

Definition at line 287 of file Row.inlines.hh.

References allocate(), copy_construct_coefficients(), flags(), Parma_Polyhedra_Library::Row_Impl_-Handler::impl, max_size(), and size().

11.176.2.6 Parma_Polyhedra_Library::Row::Row (const Row & *y*, dimension_type *sz*, dimension_type *capacity*) [inline]

Copy constructor with specified size and capacity.

It is assumed that `sz` is greater than or equal to the size of `y` and, of course, that `sz` is less than or equal to `capacity`.

Definition at line 297 of file Row.inlines.hh.

References allocate(), copy_construct_coefficients(), expand_within_capacity(), flags(), Parma_Polyhedra_Library::Row_Impl_Handler::impl, max_size(), and size().

11.176.2.7 Parma_Polyhedra_Library::Row::~Row () [inline]

Destructor.

Definition at line 309 of file Row.inlines.hh.

11.176.3 Member Function Documentation

11.176.3.1 void Parma_Polyhedra_Library::Row::allocate (dimension_type *capacity*, Flags *f*) [inline]

Allocates memory for a default constructed Row object, setting flags to `f` and allowing for `capacity` coefficients at most.

It is assumed that no allocation has been performed before (otherwise, a memory leak will occur). After execution, the size of the Row object is zero.

Definition at line 212 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl, and max_size().

Referenced by construct(), and Row().

11.176.3.2 void Parma_Polyhedra_Library::Row::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row, Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 162 of file Row.cc.

References Parma_Polyhedra_Library::Row::Flags::ascii_dump(), flags(), and size().

11.176.3.3 void Parma_Polyhedra_Library::Row::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row, Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_

Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_-Library::PIP_Tree_Node::Artificial_Parameter.

Referenced by Parma_Polyhedra_Library::MIP_Problem::ascii_dump().

11.176.3.4 bool Parma_Polyhedra_Library::Row::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Reimplemented in Parma_Polyhedra_Library::Linear_Row, Parma_Polyhedra_Library::Linear_-Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_-Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_-Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 176 of file Row.cc.

References shrink(), size(), and swap().

11.176.3.5 void Parma_Polyhedra_Library::Row::assign (Row & *y*) [inline]

Assigns the implementation of `y` to `*this`.

To be used with extra care, since it may easily cause memory leaks or undefined behavior.

Definition at line 327 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl.

11.176.3.6 void Parma_Polyhedra_Library::Row::construct (dimension_type *sz*, dimension_type *capacity*, Flags *f*) [inline]

Constructs properly a default-constructed element.

Parameters

sz The size of the row that will be constructed;

capacity The capacity of the row that will be constructed;

f Flags for the row that will be constructed.

The row that is constructed has storage for `capacity` elements, `sz` of which are default-constructed now. The row flags are set to `f`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row.

Definition at line 250 of file Row.inlines.hh.

References allocate(), expand_within_capacity(), and max_size().

11.176.3.7 void Parma_Polyhedra_Library::Row::construct (dimension_type *sz*, Flags *f*) [inline]

Constructs properly a default-constructed element.

Builds a row with size and capacity `sz` and flags `f`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row.

Definition at line 259 of file Row.inlines.hh.

Referenced by Row().

11.176.3.8 void Parma_Polyhedra_Library::Row::copy_construct_coefficients (const Row & *y*) [inline, private]

Exception-safe copy construction mechanism for coefficients.

Definition at line 241 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::Impl::copy_construct_coefficients(), Parma_Polyhedra_Library::Row_Impl_Handler::impl, and size().

Referenced by Row().

11.176.3.9 void Parma_Polyhedra_Library::Row::expand_within_capacity (dimension_type *new_size*) [inline]

Expands the row to size `new_size`.

Adds new positions to the implementation of the row obtaining a new row with size `new_size`. It is assumed that `new_size` is between the current size and capacity of the row.

Definition at line 232 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::Impl::expand_within_capacity(), and Parma_Polyhedra_Library::Row_Impl_Handler::impl.

Referenced by construct(), and Row().

11.176.3.10 memory_size_type Parma_Polyhedra_Library::Row::external_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the size in bytes of the memory managed by `*this`, provided the capacity of `*this` is given by `capacity`.

Definition at line 357 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl, and Parma_Polyhedra_Library::Row_Impl_Handler::Impl::total_memory_in_bytes().

11.176.3.11 memory_size_type Parma_Polyhedra_Library::Row::external_memory_in_bytes () const [inline]

Returns a lower bound to the size in bytes of the memory managed by `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 367 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl, and Parma_Polyhedra_Library::Row_Impl_Handler::Impl::total_memory_in_bytes().

Referenced by Parma_Polyhedra_Library::MIP_Problem::external_memory_in_bytes(), and total_memory_in_bytes().

11.176.3.12 Row::Flags & Parma_Polyhedra_Library::Row::flags () [inline]

Returns a non-const reference to the flags of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row.

Definition at line 182 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::Impl::flags(), and Parma_Polyhedra_Library::Row_Impl_Handler::impl.

11.176.3.13 const Row::Flags & Parma_Polyhedra_Library::Row::flags () const [inline]

Returns a const reference to the flags of `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row.

Definition at line 177 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::Impl::flags(), and Parma_Polyhedra_Library::Row_Impl_Handler::impl.

Referenced by ascii_dump(), operator==(), and Row().

11.176.3.14 dimension_type Parma_Polyhedra_Library::Row::max_size () [inline, static]

Returns the size() of the largest possible Row.

Definition at line 167 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::Impl::max_size().

Referenced by allocate(), construct(), Parma_Polyhedra_Library::Matrix::max_num_columns(), Parma_Polyhedra_Library::Linear_Row::max_space_dimension(), Parma_Polyhedra_Library::Congruence::max_space_dimension(), OK(), and Row().

11.176.3.15 void Parma_Polyhedra_Library::Row::normalize ()

Normalizes the modulo of coefficients so that they are mutually prime.

Computes the Greatest Common Divisor (GCD) among the elements of the row and normalizes them by the GCD itself.

Reimplemented in Parma_Polyhedra_Library::Congruence.

Definition at line 87 of file Row.cc.

References Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::neg_assign(), PPL_DIRTY_TEMP_COEFFICIENT, and size().

Referenced by Parma_Polyhedra_Library::Generator_System::add_corresponding_closure_points(), Parma_Polyhedra_Library::Polyhedron::add_generator(), Parma_Polyhedra_Library::Polyhedron::BFT00_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Generator::is_equivalent_to(), Parma_Polyhedra_Library::Constraint::is_equivalent_to(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), Parma_Polyhedra_Library::Linear_Row::strong_normalize(), Parma_Polyhedra_Library::Polyhedron::strongly_minimize_generators(), Parma_Polyhedra_Library::Polyhedron::time_elapse_assign(), and Parma_Polyhedra_Library::Polyhedron::topological_closure_assign().

11.176.3.16 bool Parma_Polyhedra_Library::Row::OK (dimension_type *row_size*, dimension_type *row_capacity*) const

Checks if all the invariants are satisfied and that the actual size and capacity match the values provided as arguments.

Reimplemented in Parma_Polyhedra_Library::Linear_Row.

Definition at line 259 of file Row.cc.

References OK(), and size().

11.176.3.17 bool Parma_Polyhedra_Library::Row::OK () const

Checks if all the invariants are satisfied.

Reimplemented in Parma_Polyhedra_Library::Linear_Row, Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 210 of file Row.cc.

References max_size(), and size().

Referenced by Parma_Polyhedra_Library::Matrix::add_recycled_row(), OK(), and Parma_Polyhedra_Library::Congruence::OK().

11.176.3.18 Row & Parma_Polyhedra_Library::Row::operator= (const Row & *y*) [inline]

Assignment operator.

Definition at line 335 of file Row.inlines.hh.

References swap().

11.176.3.19 Coefficient_traits::const_reference Parma_Polyhedra_Library::Row::operator[ ] (dimension_type *k*) const [inline]

Returns a constant reference to the element of the row indexed by `k`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 351 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl.

11.176.3.20 Coefficient & Parma_Polyhedra_Library::Row::operator[ ] (dimension_type *k*) [inline]

Returns a reference to the element of the row indexed by `k`.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 345 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl.

Referenced by Parma_Polyhedra_Library::Congruence::is_equal_at_dimension().

11.176.3.21 void Parma_Polyhedra_Library::Row::print () const

Prints `*this` to `std::cerr` using `operator<<`.

Reimplemented in Parma_Polyhedra_Library::Linear_Row, Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

11.176.3.22 void Parma_Polyhedra_Library::Row::shrink (dimension_type *new_size*) [inline]

Shrinks the row by erasing elements at the end.

Destroys elements of the row implementation from position `new_size` to the end. It is assumed that `new_size` is not greater than the current size.

Definition at line 313 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl, and Parma_Polyhedra_Library::Row_-Impl_Handler::Impl::shrink().

Referenced by ascii_load(), Parma_Polyhedra_Library::Linear_Row::ascii_load(), Parma_Polyhedra_-Library::Grid_Generator::ascii_load(), Parma_Polyhedra_Library::Congruence::ascii_load(), and Parma_-Polyhedra_Library::MIP_Problem::erase_artificials().

11.176.3.23 dimension_type Parma_Polyhedra_Library::Row::size () const [inline]

Gives the number of coefficients currently in use.

Reimplemented in Parma_Polyhedra_Library::Grid_Generator.

Definition at line 172 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl, and Parma_Polyhedra_Library::Row_-Impl_Handler::Impl::size().

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint(), Parma_Polyhedra_-Library::Linear_System::add_pending_row(), Parma_Polyhedra_Library::Linear_System::add_-row(), Parma_Polyhedra_Library::Constraint_System::affine_preimage(), Parma_Polyhedra_-Library::Congruence_System::affine_preimage(), Parma_Polyhedra_Library::Linear_Row::all_-homogeneous_terms_are_zero(), ascii_dump(), Parma_Polyhedra_Library::MIP_Problem::ascii_dump(), Parma_Polyhedra_Library::Linear_Row::ascii_dump(), Parma_Polyhedra_Library::Congruence::ascii_-dump(), ascii_load(), Parma_Polyhedra_Library::Linear_Row::ascii_load(), Parma_Polyhedra_-Library::Grid_Generator::ascii_load(), Parma_Polyhedra_Library::Congruence::ascii_load(), Parma_-Polyhedra_Library::Scalar_Products::assign(), Parma_Polyhedra_Library::Polyhedron::BFT00_-poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Linear_Row::compare(), Parma_-Polyhedra_Library::MIP_Problem::compute_simplex_using_exact_pricing(), Parma_Polyhedra_-Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_-Library::Congruence::Congruence(), copy_construct_coefficients(), Parma_Polyhedra_Library::MIP_-Problem::erase_artificials(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign(), Parma_Polyhedra_Library::Linear_System::insert(), Parma_Polyhedra_Library::Linear_System::insert_-pending(), Parma_Polyhedra_Library::Congruence_System::insert_verbatim(), Parma_Polyhedra_-Library::Constraint::is_inconsistent(), Parma_Polyhedra_Library::Constraint::is_tautological(), Parma_-Polyhedra_Library::Linear_Row::is_zero(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), Parma_Polyhedra_Library::Linear_Row::linear_combine(), Parma_Polyhedra_Library::Linear_-Expression::Linear_Expression(), Parma_Polyhedra_Library::Congruence::modulus(), normalize(), Parma_Polyhedra_Library::Congruence::normalize(), Parma_Polyhedra_Library::Congruence_-System::normalize_moduli(), OK(), Parma_Polyhedra_Library::MIP_Problem::OK(), Parma_-Polyhedra_Library::Congruence::operator/=(), operator==(), Parma_Polyhedra_Library::MIP_-Problem::process_pending_constraints(), Parma_Polyhedra_Library::Scalar_Products::reduced_assign(), Row(), Parma_Polyhedra_Library::PIP_Solution_Node::row_sign(), Parma_Polyhedra_Library::MIP_-Problem::second_phase(), Parma_Polyhedra_Library::Congruence::set_is_equality(), Parma_Polyhedra_-Library::Linear_Row::sign_normalize(), Parma_Polyhedra_Library::Congruence::sign_normalize(), Parma_Polyhedra_Library::Linear_Row::space_dimension(), Parma_Polyhedra_Library::Linear_-Expression::space_dimension(), Parma_Polyhedra_Library::Congruence::space_dimension(), Parma_-Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index(), Parma_Polyhedra_-Library::MIP_Problem::steepest_edge_float_entering_index(), Parma_Polyhedra_Library::MIP_-Problem::textbook_entering_index(), Parma_Polyhedra_Library::Generator::type(), and Parma_-Polyhedra_Library::Constraint::type().

11.176.3.24 void Parma_Polyhedra_Library::Row::swap (Row & *y*) [inline]

Swaps `*this` with `y`.

Definition at line 319 of file Row.inlines.hh.

References Parma_Polyhedra_Library::Row_Impl_Handler::impl.

Referenced by ascii_load(), Parma_Polyhedra_Library::Linear_Row::ascii_load(), Parma_Polyhedra_Library::Grid_Generator::ascii_load(), Parma_Polyhedra_Library::Congruence::ascii_load(), Parma_Polyhedra_Library::MIP_Problem::erase_artificials(), operator=(), Parma_Polyhedra_Library::MIP_Problem::second_phase(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and swap().

11.176.3.25 memory_size_type Parma_Polyhedra_Library::Row::total_memory_in_bytes (dimension_type *capacity*) const [inline]

Returns the total size in bytes of the memory occupied by `*this`, provided the capacity of `*this` is given by `capacity`.

Definition at line 362 of file Row.inlines.hh.

References external_memory_in_bytes().

11.176.3.26 memory_size_type Parma_Polyhedra_Library::Row::total_memory_in_bytes () const [inline]

Returns a lower bound to the total size in bytes of the memory occupied by `*this`.

Reimplemented in Parma_Polyhedra_Library::Linear_Expression, Parma_Polyhedra_Library::Constraint, Parma_Polyhedra_Library::Congruence, Parma_Polyhedra_Library::Generator, Parma_Polyhedra_Library::Grid_Generator, and Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter.

Definition at line 376 of file Row.inlines.hh.

References external_memory_in_bytes().

11.176.4 Friends And Related Function Documentation

11.176.4.1 void iter_swap (std::vector< Parma_Polyhedra_Library::Row >::iterator *x*, std::vector< Parma_Polyhedra_Library::Row >::iterator *y*) [related]

Specializes `std::iter_swap`.

Definition at line 399 of file Row.inlines.hh.

References Parma_Polyhedra_Library::swap().

11.176.4.2 bool operator!= (const Row & *x*, const Row & *y*) [related]

Returns `true` if and only if `x` and `y` are different.

Definition at line 382 of file Row.inlines.hh.

11.176.4.3 bool operator== (const Row & *x*, const Row & *y*) [related]

Returns `true` if and only if `x` and `y` are equal.

Definition at line 302 of file Row.cc.

References flags(), and size().

11.176.4.4 void swap (Parma_Polyhedra_Library::Row & *x*, Parma_Polyhedra_Library::Row & *y*) [related]

Specializes `std::swap`.

Definition at line 393 of file Row.inlines.hh.

References swap().

The documentation for this class was generated from the following files:

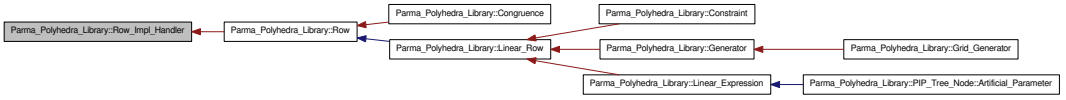
- Row.defs.hh
- Row.inlines.hh
- Row.cc

11.177 Parma_Polyhedra_Library::Row_Impl_Handler Class Reference

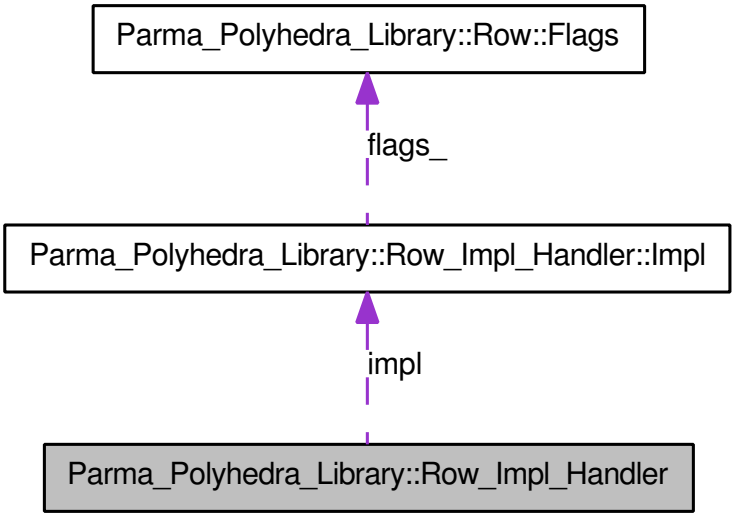
The handler of the actual Row implementation.

`#include <Row.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Row_Impl_Handler:



Collaboration diagram for Parma_Polyhedra_Library::Row_Impl_Handler:



Classes

- class Impl

  The actual implementation of a Row object.

Public Member Functions

- Row_Impl_Handler ()

  Default constructor.

- ~Row_Impl_Handler ()

  Destructor.

Public Attributes

- Impl * impl

  A pointer to the actual implementation.

Private Member Functions

- Row_Impl_Handler (const Row_Impl_Handler &)

  Private and unimplemented: copy construction is not allowed.

- Row_Impl_Handler & operator= (const Row_Impl_Handler &)

  Private and unimplemented: copy assignment is not allowed.

11.177.1 Detailed Description

The handler of the actual Row implementation. Exception-safety is the only responsibility of this class: it has to ensure that its `impl` member is correctly deallocated.

Definition at line 59 of file Row.defs.hh.

11.177.2 Constructor & Destructor Documentation

11.177.2.1 Parma_Polyhedra_Library::Row_Impl_Handler::Row_Impl_Handler () [inline]

Default constructor.

Definition at line 194 of file Row.inlines.hh.

11.177.2.2 Parma_Polyhedra_Library::Row_Impl_Handler::~Row_Impl_Handler () [inline]

Destructor.

Definition at line 202 of file Row.inlines.hh.

References impl.

11.177.2.3 Parma_Polyhedra_Library::Row_Impl_Handler::Row_Impl_Handler (const Row_Impl_Handler &) [private]

Private and unimplemented: copy construction is not allowed.

11.177.3 Member Function Documentation

11.177.3.1 Row_Impl_Handler& Parma_Polyhedra_Library::Row_Impl_Handler::operator= (const Row_Impl_Handler &) [private]

Private and unimplemented: copy assignment is not allowed.

11.177.4 Member Data Documentation

11.177.4.1 Impl* Parma_Polyhedra_Library::Row_Impl_Handler::impl

A pointer to the actual implementation.

Definition at line 67 of file Row.defs.hh.

Referenced by Parma_Polyhedra_Library::Row::allocate(), Parma_Polyhedra_Library::Row::assign(), Parma_Polyhedra_Library::Row::copy_construct_coefficients(), Parma_Polyhedra_Library::Row::expand_within_capacity(), Parma_Polyhedra_Library::Row::external_memory_in_bytes(), Parma_Polyhedra_Library::Row::flags(), Parma_Polyhedra_Library::Row::operator[ ](), Parma_Polyhedra_Library::Row::Row(), Parma_Polyhedra_Library::Row::shrink(), Parma_Polyhedra_Library::Row::size(), Parma_Polyhedra_Library::Row::swap(), and ~Row_Impl_Handler().

The documentation for this class was generated from the following files:

- Row.defs.hh
- Row.inlines.hh

11.178 Parma_Polyhedra_Library::Linear_System::Row_Less_Than Struct Reference

Ordering predicate (used when implementing the sort algorithm).

Public Member Functions

- bool operator() (const Row &x, const Row &y) const

11.178.1 Detailed Description

Ordering predicate (used when implementing the sort algorithm).

Definition at line 394 of file Linear_System.defs.hh.

11.178.2 Member Function Documentation

11.178.2.1 bool Parma_Polyhedra_Library::Linear_System::Row_Less_Than::operator() (const Row & *x*, const Row & *y*) const [inline]

Definition at line 232 of file Linear_System.inlines.hh.

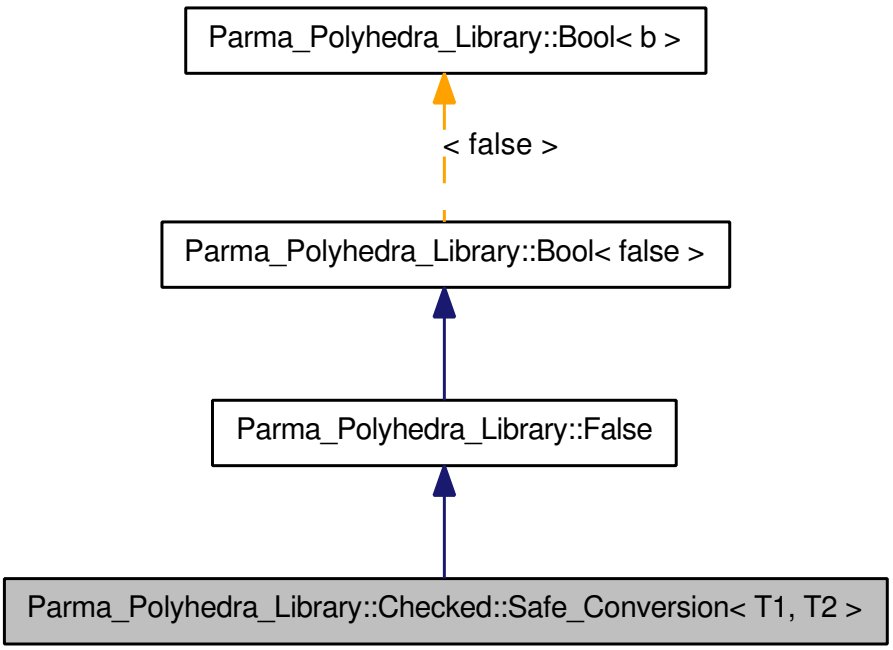
The documentation for this struct was generated from the following files:

- Linear_System.defs.hh
- Linear_System.inlines.hh

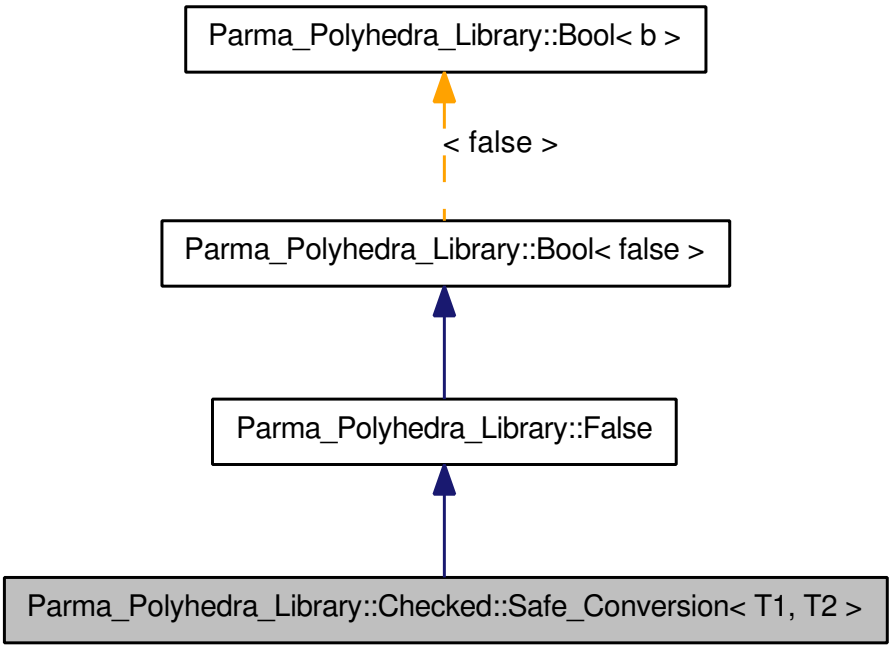
11.179 Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 > Struct Template Reference

`#include <checked.inlines.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 >:



Collaboration diagram for Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 >:



11.179.1 Detailed Description

template<typename T1, typename T2> struct Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 >

Definition at line 46 of file checked.inlines.hh.

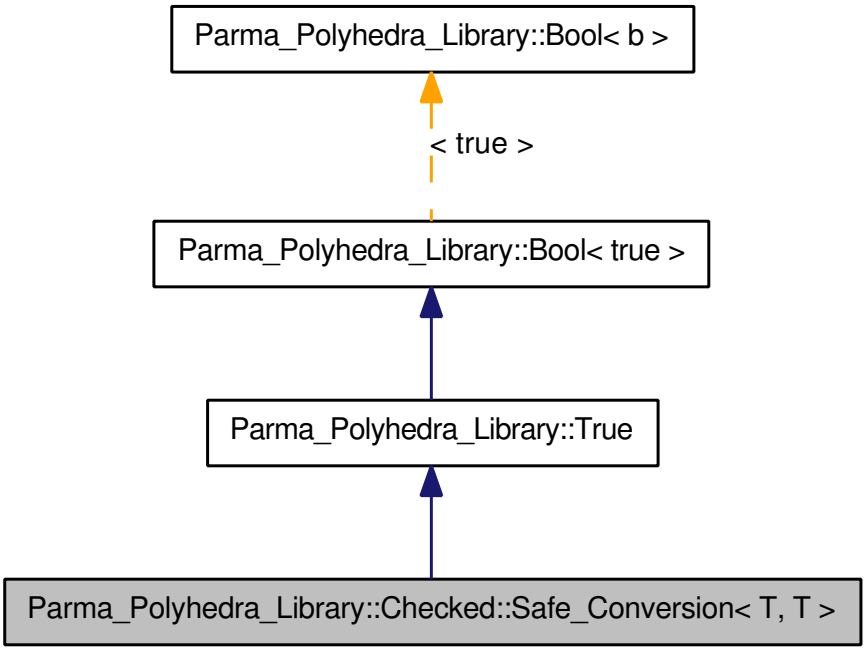
The documentation for this struct was generated from the following file:

- checked.inlines.hh

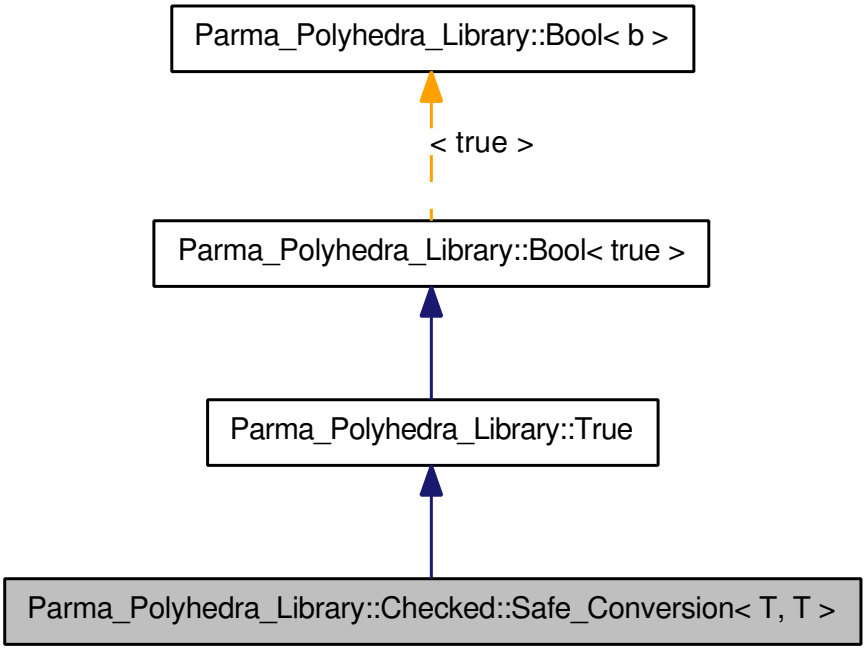
11.180 Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T > Struct Template Reference

`#include <checked.inlines.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T >:



Collaboration diagram for Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T >:



11.180.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T >

Definition at line 49 of file checked.inlines.hh.

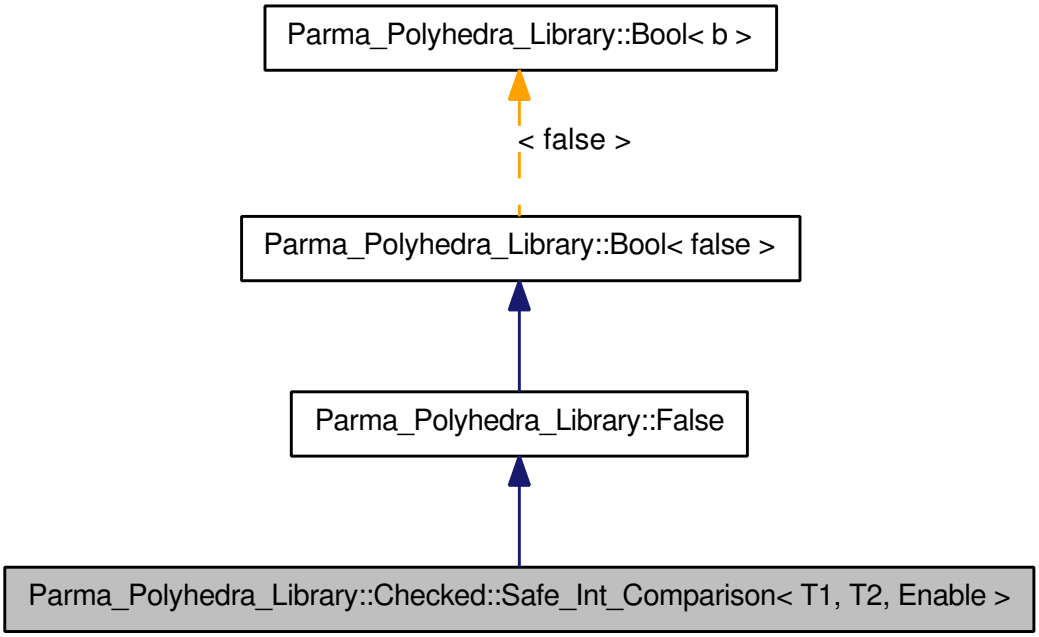
The documentation for this struct was generated from the following file:

- checked.inlines.hh

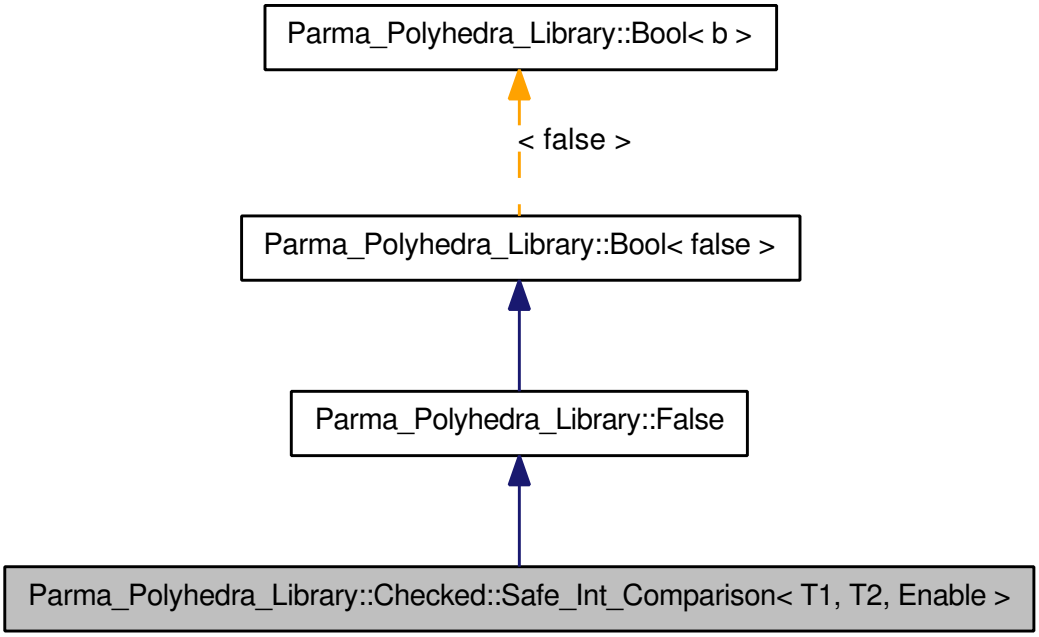
11.181 Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable > Struct Template Reference

`#include <checked.inlines.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable >:



Collaboration diagram for Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable >:



11.181.1 Detailed Description

template<typename T1, typename T2, typename Enable = void> struct Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable >

Definition at line 443 of file checked.inlines.hh.

The documentation for this struct was generated from the following file:

- checked.inlines.hh

11.182 Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type > Struct Template Reference

`#include <checked.inlines.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >:

Collaboration diagram for Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >:

11.182.1 Detailed Description

template<typename T1, typename T2> struct Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >

Definition at line 447 of file checked.inlines.hh.

The documentation for this struct was generated from the following file:

- checked.inlines.hh

11.183 Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy Struct Reference

`#include <Interval.inlines.hh>`

Public Member Functions

- const_bool_nodef (may_be_empty, true)
- const_bool_nodef (may_contain_infinity, true)
- const_bool_nodef (check_inexact, false)

11.183.1 Detailed Description

Definition at line 95 of file Interval.inlines.hh.

11.183.2 Member Function Documentation

11.183.2.1 Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy::const_bool_nodef (check_inexact, false)

11.183.2.2 Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy::const_bool_nodef (may_contain_infinity, true)

11.183.2.3 Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy::const_bool_nodef (may_be_empty, true)

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.184 Parma_Polyhedra_Library::Scalar_Products Class Reference

A class implementing various scalar product functions.

`#include <Scalar_Products.defs.hh>`

Static Public Member Functions

- static void assign (Coefficient &z, const Linear_Row &x, const Linear_Row &y)
  Computes the scalar product of `x` and `y` and assigns it to `z`.
- static void assign (Coefficient &z, const Constraint &c, const Generator &g)
  Computes the scalar product of `c` and `g` and assigns it to `z`.
- static void assign (Coefficient &z, const Generator &g, const Constraint &c)
  Computes the scalar product of `g` and `c` and assigns it to `z`.
- static void assign (Coefficient &z, const Constraint &c, const Grid_Generator &g)
  Computes the scalar product of `c` and `g` and assigns it to `z`.
- static void assign (Coefficient &z, const Grid_Generator &g, const Congruence &cg)
  Computes the scalar product of `g` and `cg` and assigns it to `z`.
- static void assign (Coefficient &z, const Congruence &cg, const Grid_Generator &g)
  Computes the scalar product of `cg` and `g` and assigns it to `z`.

- static int sign (const Linear_Row &x, const Linear_Row &y)

  Returns the sign of the scalar product between x *and* y.

- static int sign (const Constraint &c, const Generator &g)

  Returns the sign of the scalar product between c *and* g.

- static int sign (const Generator &g, const Constraint &c)

  Returns the sign of the scalar product between g *and* c.

- static int sign (const Constraint &c, const Grid_Generator &g)

  Returns the sign of the scalar product between c *and* g.

- static void reduced_assign (Coefficient &z, const Linear_Row &x, const Linear_Row &y)

  Computes the reduced scalar product of x *and* y*, where the* ϵ *coefficient of* x *is ignored, and assigns the result to* z.

- static void reduced_assign (Coefficient &z, const Constraint &c, const Generator &g)

  Computes the reduced scalar product of c *and* g*, where the* ϵ *coefficient of* c *is ignored, and assigns the result to* z.

- static void reduced_assign (Coefficient &z, const Generator &g, const Constraint &c)

  Computes the reduced scalar product of g *and* c*, where the* ϵ *coefficient of* g *is ignored, and assigns the result to* z.

- static void reduced_assign (Coefficient &z, const Grid_Generator &g, const Congruence &cg)

  Computes the reduced scalar product of g *and* cg*, where the* ϵ *coefficient of* g *is ignored, and assigns the result to* z.

- static int reduced_sign (const Linear_Row &x, const Linear_Row &y)

  Returns the sign of the reduced scalar product of x *and* y*, where the* ϵ *coefficient of* x *is ignored.*

- static int reduced_sign (const Constraint &c, const Generator &g)

  Returns the sign of the reduced scalar product of c *and* g*, where the* ϵ *coefficient of* c *is ignored.*

- static int reduced_sign (const Generator &g, const Constraint &c)

  Returns the sign of the reduced scalar product of g *and* c*, where the* ϵ *coefficient of* g *is ignored.*

- static void homogeneous_assign (Coefficient &z, const Linear_Row &x, const Linear_Row &y)

  Computes the homogeneous scalar product of x *and* y*, where the inhomogeneous terms are ignored, and assigns the result to* z.

- static void homogeneous_assign (Coefficient &z, const Linear_Expression &e, const Generator &g)

  Computes the homogeneous scalar product of e *and* g*, where the inhomogeneous terms are ignored, and assigns the result to* z.

- static void homogeneous_assign (Coefficient &z, const Grid_Generator &g, const Constraint &c)

  Computes the homogeneous scalar product of g *and* c*, where the inhomogeneous terms are ignored, and assigns the result to* z.

- static void homogeneous_assign (Coefficient &z, const Grid_Generator &g, const Congruence &cg)

  Computes the homogeneous scalar product of `g` *and* `cg`*, where the inhomogeneous terms are ignored, and assigns the result to* `z`*.*

- static void homogeneous_assign (Coefficient &z, const Linear_Expression &e, const Grid_Generator &g)

  Computes the homogeneous scalar product of `e` *and* `g`*, where the inhomogeneous terms are ignored, and assigns the result to* `z`*.*

- static int homogeneous_sign (const Linear_Row &x, const Linear_Row &y)

  Returns the sign of the homogeneous scalar product of `x` *and* `y`*, where the inhomogeneous terms are ignored.*

- static int homogeneous_sign (const Linear_Expression &e, const Generator &g)

  Returns the sign of the homogeneous scalar product of `e` *and* `g`*, where the inhomogeneous terms are ignored.*

- static int homogeneous_sign (const Linear_Expression &e, const Grid_Generator &g)

  Returns the sign of the homogeneous scalar product of `e` *and* `g`*, where the inhomogeneous terms are ignored,.*

- static int homogeneous_sign (const Grid_Generator &g, const Constraint &c)

  Returns the sign of the homogeneous scalar product of `g` *and* `c`*, where the inhomogeneous terms are ignored,.*

11.184.1 Detailed Description

A class implementing various scalar product functions. When computing the scalar product of (Linear_Row or Constraint or Generator) objects `x` and `y`, it is assumed that the space dimension of the first object `x` is less than or equal to the space dimension of the second object `y`.

Definition at line 45 of file Scalar_Products.defs.hh.

11.184.2 Member Function Documentation

11.184.2.1 void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Congruence & *cg*, const Grid_Generator & *g*) [static]

Computes the scalar product of `cg` and `g` and assigns it to `z`.

Definition at line 82 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Grid_Generator::size(), and Parma_Polyhedra_Library::Row::size().

11.184.2.2 void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Grid_Generator & *g*, const Congruence & *cg*) [static]

Computes the scalar product of `g` and `cg` and assigns it to `z`.

Definition at line 56 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Grid_Generator::size().

11.184.2.3 void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Constraint & *c*, const Grid_Generator & *g*) [static]

Computes the scalar product of `c` and `g` and assigns it to `z`.

Definition at line 69 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Grid_-Generator::size(), and Parma_Polyhedra_Library::Row::size().

11.184.2.4 static void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Generator & *g*, const Constraint & *c*) [static]

Computes the scalar product of `g` and `c` and assigns it to `z`.

11.184.2.5 void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Constraint & *c*, const Generator & *g*) [static]

Computes the scalar product of `c` and `g` and assigns it to `z`.

Definition at line 44 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), and Parma_Polyhedra_Library::Row::size().

11.184.2.6 void Parma_Polyhedra_Library::Scalar_Products::assign (Coefficient & *z*, const Linear_Row & *x*, const Linear_Row & *y*) [static]

Computes the scalar product of `x` and `y` and assigns it to `z`.

Definition at line 32 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Grid_Generator_System::affine_image(), Parma_Polyhedra_-Library::Generator_System::affine_image(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Polyhedron::relation_with(), Parma_Polyhedra_Library::Grid::relation_-with(), Parma_Polyhedra_Library::Congruence_System::satisfies_all_congruences(), sign(), and Parma_-Polyhedra_Library::Grid::simplify_using_context_assign().

11.184.2.7 void Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign (Coefficient & *z*, const Linear_Expression & *e*, const Grid_Generator & *g*) [inline, static]

Computes the *homogeneous* scalar product of `e` and `g`, where the inhomogeneous terms are ignored, and assigns the result to `z`.

Definition at line 98 of file Scalar_Products.inlines.hh.

References homogeneous_assign().

11.184.2.8 void Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign (Coefficient & *z*, const Grid_Generator & *g*, const Congruence & *cg*) [static]

Computes the *homogeneous* scalar product of `g` and `cg`, where the inhomogeneous terms are ignored, and assigns the result to `z`.

Definition at line 136 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Grid_Generator::size().

11.184.2.9 void Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign (Coefficient & *z*, const Grid_Generator & *g*, const Constraint & *c*) [static]

Computes the *homogeneous* scalar product of `g` and `c`, where the inhomogeneous terms are ignored, and assigns the result to `z`.

Definition at line 150 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Grid_Generator::size().

11.184.2.10 void Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign (Coefficient & *z*, const Linear_Expression & *e*, const Generator & *g*) [inline, static]

Computes the *homogeneous* scalar product of `e` and `g`, where the inhomogeneous terms are ignored, and assigns the result to `z`.

Definition at line 89 of file Scalar_Products.inlines.hh.

References homogeneous_assign().

11.184.2.11 void Parma_Polyhedra_Library::Scalar_Products::homogeneous_assign (Coefficient & *z*, const Linear_Row & *x*, const Linear_Row & *y*) [static]

Computes the *homogeneous* scalar product of `x` and `y`, where the inhomogeneous terms are ignored, and assigns the result to `z`.

Definition at line 122 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), and Parma_Polyhedra_Library::Row::size().

Referenced by Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Grid::frequency_no_check(), homogeneous_assign(), homogeneous_sign(), Parma_Polyhedra_-

Library::Polyhedron::max_min(), and Parma_Polyhedra_Library::Grid::max_min().

11.184.2.12 int Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign (const Grid_Generator & *g*, const Constraint & *c*) [inline, static]

Returns the sign of the *homogeneous* scalar product of `g` and `c`, where the inhomogeneous terms are ignored,.

Definition at line 121 of file Scalar_Products.inlines.hh.

References homogeneous_assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

11.184.2.13 int Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign (const Linear_Expression & *e*, const Grid_Generator & *g*) [inline, static]

Returns the sign of the *homogeneous* scalar product of `e` and `g`, where the inhomogeneous terms are ignored,.

Definition at line 114 of file Scalar_Products.inlines.hh.

References homogeneous_sign().

11.184.2.14 int Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign (const Linear_Expression & *e*, const Generator & *g*) [inline, static]

Returns the sign of the *homogeneous* scalar product of `e` and `g`, where the inhomogeneous terms are ignored.

Definition at line 107 of file Scalar_Products.inlines.hh.

References homogeneous_sign().

11.184.2.15 int Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign (const Linear_Row & *x*, const Linear_Row & *y*) [inline, static]

Returns the sign of the *homogeneous* scalar product of `x` and `y`, where the inhomogeneous terms are ignored.

Definition at line 51 of file Scalar_Products.inlines.hh.

References homogeneous_assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

Referenced by Parma_Polyhedra_Library::Polyhedron::bounds(), Parma_Polyhedra_Library::Grid::bounds_no_check(), and homogeneous_sign().

11.184.2.16 void Parma_Polyhedra_Library::Scalar_Products::reduced_assign (Coefficient & *z*, const Grid_Generator & *g*, const Congruence & *cg*) [static]

Computes the *reduced* scalar product of g and cg, where the ϵ coefficient of g is ignored, and assigns the result to z.

Definition at line 109 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Row::size(), and Parma_Polyhedra_Library::Grid_Generator::size().

11.184.2.17 static void Parma_Polyhedra_Library::Scalar_Products::reduced_assign (Coefficient & *z*, const Generator & *g*, const Constraint & *c*) [static]

Computes the *reduced* scalar product of g and c, where the ϵ coefficient of g is ignored, and assigns the result to z.

11.184.2.18 static void Parma_Polyhedra_Library::Scalar_Products::reduced_assign (Coefficient & *z*, const Constraint & *c*, const Generator & *g*) [static]

Computes the *reduced* scalar product of c and g, where the ϵ coefficient of c is ignored, and assigns the result to z.

11.184.2.19 void Parma_Polyhedra_Library::Scalar_Products::reduced_assign (Coefficient & *z*, const Linear_Row & *x*, const Linear_Row & *y*) [static]

Computes the *reduced* scalar product of x and y, where the ϵ coefficient of x is ignored, and assigns the result to z.

Definition at line 95 of file Scalar_Products.cc.

References Parma_Polyhedra_Library::add_mul_assign(), Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), and Parma_Polyhedra_Library::Row::size().

Referenced by reduced_sign().

11.184.2.20 int Parma_Polyhedra_Library::Scalar_Products::reduced_sign (const Generator & *g*, const Constraint & *c*) [inline, static]

Returns the sign of the *reduced* scalar product of g and c, where the ϵ coefficient of g is ignored.

Definition at line 83 of file Scalar_Products.inlines.hh.

References reduced_sign().

11.184.2.21 int Parma_Polyhedra_Library::Scalar_Products::reduced_sign (const Constraint & *c*, const Generator & *g*) [inline, static]

Returns the sign of the *reduced* scalar product of c and g, where the ϵ coefficient of c is ignored.

Definition at line 77 of file Scalar_Products.inlines.hh.

References reduced_sign().

11.184.2.22 int Parma_Polyhedra_Library::Scalar_Products::reduced_sign (const Linear_Row & *x*, const Linear_Row & *y*) [inline, static]

Returns the sign of the *reduced* scalar product of `x` and `y`, where the ϵ coefficient of `x` is ignored.

Definition at line 44 of file Scalar_Products.inlines.hh.

References PPL_DIRTY_TEMP_COEFFICIENT, and reduced_assign().

Referenced by Parma_Polyhedra_Library::Polyhedron::BHZ09_NNC_poly_hull_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Sign::operator()(), reduced_sign(), and Parma_Polyhedra_Library::Generator_-System::relation_with().

11.184.2.23 int Parma_Polyhedra_Library::Scalar_Products::sign (const Constraint & *c*, const Grid_Generator & *g*) [inline, static]

Returns the sign of the scalar product between `c` and `g`.

Definition at line 70 of file Scalar_Products.inlines.hh.

References assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

11.184.2.24 int Parma_Polyhedra_Library::Scalar_Products::sign (const Generator & *g*, const Constraint & *c*) [inline, static]

Returns the sign of the scalar product between `g` and `c`.

Definition at line 64 of file Scalar_Products.inlines.hh.

References sign().

11.184.2.25 int Parma_Polyhedra_Library::Scalar_Products::sign (const Constraint & *c*, const Generator & *g*) [inline, static]

Returns the sign of the scalar product between `c` and `g`.

Definition at line 58 of file Scalar_Products.inlines.hh.

References sign().

11.184.2.26 int Parma_Polyhedra_Library::Scalar_Products::sign (const Linear_Row & *x*, const Linear_Row & *y*) [inline, static]

Returns the sign of the scalar product between x and y.

Definition at line 37 of file Scalar_Products.inlines.hh.

References assign(), and PPL_DIRTY_TEMP_COEFFICIENT.

Referenced by Parma_Polyhedra_Library::Polyhedron::BHRZ03_combining_constraints(), Parma_Polyhedra_Library::Polyhedron::is_included_in(), Parma_Polyhedra_Library::MIP_Problem::is_satisfied(), Parma_Polyhedra_Library::MIP_Problem::is_saturated(), Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator()(), Parma_Polyhedra_Library::Grid::relation_with(), Parma_Polyhedra_Library::Generator_System::relation_with(), Parma_Polyhedra_Library::Polyhedron::select_H79_constraints(), sign(), Parma_Polyhedra_Library::Polyhedron::simplify_using_context_assign(), Parma_Polyhedra_Library::Polyhedron::update_sat_c(), and Parma_Polyhedra_Library::Polyhedron::update_sat_g().

The documentation for this class was generated from the following files:

- Scalar_Products.defs.hh
- Scalar_Products.inlines.hh
- Scalar_Products.cc

11.185 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef Interval_Boundary_Type type

11.185.1 Detailed Description

template<typename Interval_Boundary_Type> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type >

Definition at line 1132 of file Interval.inlines.hh.

11.185.2 Member Typedef Documentation

11.185.2.1 template<typename Interval_Boundary_Type> typedef Interval_Boundary_Type Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type >::type

Definition at line 1133 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.186 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.186.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char >

Definition at line 1142 of file Interval.inlines.hh.

11.186.2 Member Typedef Documentation

11.186.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char >::type

Definition at line 1143 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.187 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef double type

11.187.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float >

Definition at line 1137 of file Interval.inlines.hh.

11.187.2 Member Typedef Documentation

11.187.2.1 typedef double Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float >::type

Definition at line 1138 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.188 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.188.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char >

Definition at line 1147 of file Interval.inlines.hh.

11.188.2 Member Typedef Documentation

11.188.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char >::type

Definition at line 1148 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.189 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.189.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int >

Definition at line 1167 of file Interval.inlines.hh.

11.189.2 Member Typedef Documentation

11.189.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int >::type

Definition at line 1168 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.190 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.190.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long >

Definition at line 1177 of file Interval.inlines.hh.

11.190.2 Member Typedef Documentation

11.190.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long >::type

Definition at line 1178 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.191 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.191.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short >

Definition at line 1157 of file Interval.inlines.hh.

11.191.2 Member Typedef Documentation

11.191.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short >::type

Definition at line 1158 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.192 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.192.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char >

Definition at line 1152 of file Interval.inlines.hh.

11.192.2 Member Typedef Documentation

11.192.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char >::type

Definition at line 1153 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.193 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.193.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int >

Definition at line 1172 of file Interval.inlines.hh.

11.193.2 Member Typedef Documentation

11.193.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int >::type

Definition at line 1173 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.194 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.194.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long >

Definition at line 1182 of file Interval.inlines.hh.

11.194.2 Member Typedef Documentation

11.194.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long >::type

Definition at line 1183 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.195 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.195.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long >

Definition at line 1187 of file Interval.inlines.hh.

11.195.2 Member Typedef Documentation

11.195.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long >::type

Definition at line 1188 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.196 Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short > Struct Template Reference

`#include <Interval.inlines.hh>`

Public Types

- typedef signed long long type

11.196.1 Detailed Description

template<> struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short >

Definition at line 1162 of file Interval.inlines.hh.

11.196.2 Member Typedef Documentation

11.196.2.1 typedef signed long long Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short >::type

Definition at line 1163 of file Interval.inlines.hh.

The documentation for this struct was generated from the following file:

- Interval.inlines.hh

11.197 Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 > Class Template Reference

This class provides the reduction method for the Shape_Preserving_Product domain.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- Shape_Preserving_Reduction ()

  Default constructor.

- void product_reduce (D1 &d1, D2 &d2)

  The congruences reduction operator for detect emptiness or any equalities implied by each of the congruences defining one of the components and the bounds of the other component. It is assumed that the components are already constraints reduced.

- ~Shape_Preserving_Reduction ()

  Destructor.

11.197.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >

This class provides the reduction method for the Shape_Preserving_Product domain. The reduction classes are used to instantiate the Partially_Reduced_Product domain.

This reduction method includes the congruences reduction. This class uses the minimized constraints defining each of the components. For each of the constraints, it checks the frequency and value for the same linear expression in the other component. If the constraint does not satisfy the implied congruence, the inhomogeneous term is adjusted so that it does. Note that unless the congruences reduction adds equalitites the shapes of the domains are unaltered.

Definition at line 243 of file Partially_Reduced_Product.defs.hh.

11.197.2 Constructor & Destructor Documentation

11.197.2.1 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::Shape_Preserving_Reduction () [inline]

Default constructor.

Definition at line 791 of file Partially_Reduced_Product.inlines.hh.

11.197.2.2 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Shape_-Preserving_Reduction< D1, D2 >::~Shape_Preserving_Reduction () [inline]

Destructor.

Definition at line 796 of file Partially_Reduced_Product.inlines.hh.

11.197.3 Member Function Documentation

11.197.3.1 template<typename D1 , typename D2 > void Parma_Polyhedra_Library::Shape_-Preserving_Reduction< D1, D2 >::product_reduce (D1 & *d1*, D2 & *d2*) [inline]

The congruences reduction operator for detect emptiness or any equalities implied by each of the congruences defining one of the components and the bounds of the other component. It is assumed that the components are already constraints reduced.

The minimized congruence system defining the domain element `d1` is used to check if `d2` intersects none, one or more than one of the hyperplanes defined by the congruences: if it intersects none, then product is set empty; if it intersects one, then the equality defining this hyperplane is added to both components; otherwise, the product is unchanged. In each case, the donor domain must provide a congruence system in minimal form.

Parameters

d1 A pointset domain element;

d2 A pointset domain element;

Definition at line 616 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::Constraint_System::begin(), Parma_Polyhedra_-Library::Constraint_System::clear(), Parma_Polyhedra_Library::Constraint_System::end(), Parma_-Polyhedra_Library::Constraint_System::insert(), Parma_Polyhedra_Library::Constraint::is_-equality(), Parma_Polyhedra_Library::Checked::le, PPL_DIRTY_TEMP_COEFFICIENT, Parma_-Polyhedra_Library::Constraints_Reduction< D1, D2 >::product_reduce(), and Parma_Polyhedra_-Library::Congruences_Reduction< D1, D2 >::product_reduce().

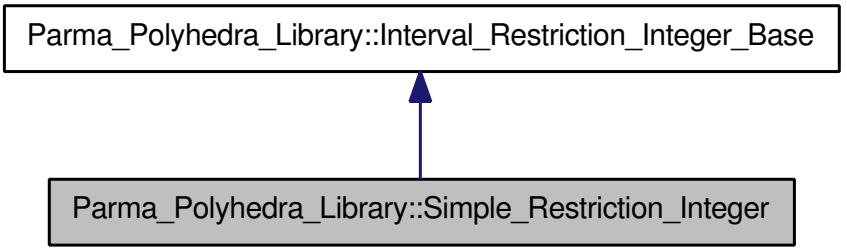
The documentation for this class was generated from the following files:

- Partially_Reduced_Product.defs.hh
- Partially_Reduced_Product.inlines.hh
- Partially_Reduced_Product.templates.hh

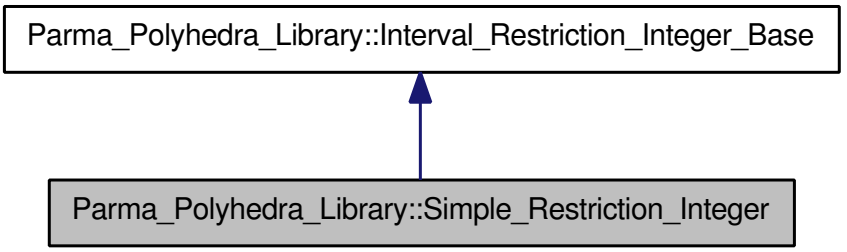
11.198 Parma_Polyhedra_Library::Simple_Restriction_Integer Class Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Simple_Restriction_Integer:



Collaboration diagram for Parma_Polyhedra_Library::Simple_Restriction_Integer:



Public Member Functions

- Simple_Restriction_Integer (bool i)
- bool get_integer () const

Private Attributes

- bool integer

11.198.1 Detailed Description

Definition at line 188 of file Interval_Restriction.defs.hh.

11.198.2 Constructor & Destructor Documentation

11.198.2.1 Parma_Polyhedra_Library::Simple_Restriction_Integer::Simple_Restriction_Integer (bool *i*) [inline]

Definition at line 190 of file Interval_Restriction.defs.hh.

11.198.3 Member Function Documentation

11.198.3.1 bool Parma_Polyhedra_Library::Simple_Restriction_Integer::get_integer () const [inline]

Definition at line 193 of file Interval_Restriction.defs.hh.

References integer.

11.198.4 Member Data Documentation

11.198.4.1 bool Parma_Polyhedra_Library::Simple_Restriction_Integer::integer [private]

Definition at line 197 of file Interval_Restriction.defs.hh.

Referenced by get_integer().

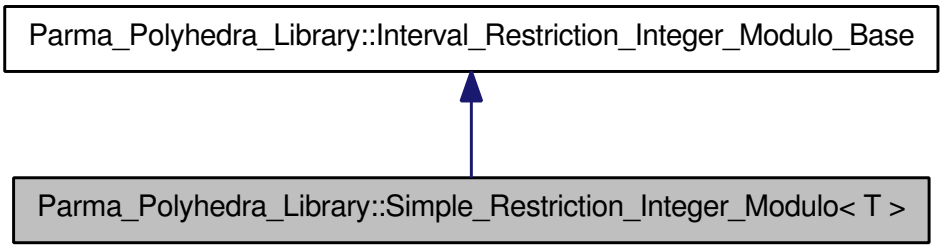
The documentation for this class was generated from the following file:

- Interval_Restriction.defs.hh

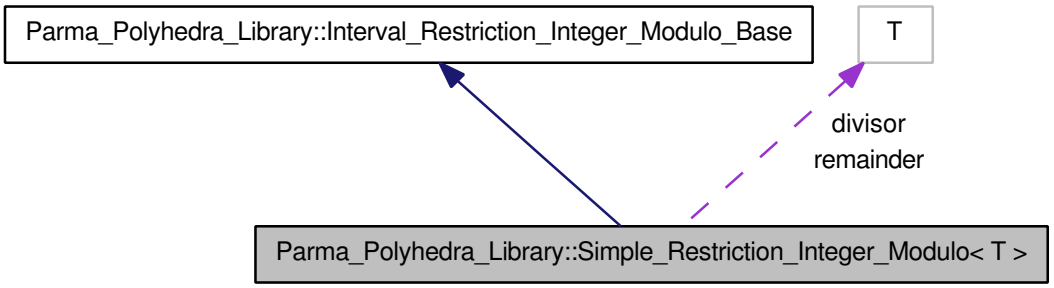
11.199 Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >:



Collaboration diagram for Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >:



Public Types

- typedef T modulo_type

Public Member Functions

- template<typename From >
  Simple_Restriction_Integer_Modulo (const From &r, const From &d)

Public Attributes

- T remainder
- T divisor

11.199.1 Detailed Description

template<typename T> struct Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >

Definition at line 415 of file Interval_Restriction.defs.hh.

11.199.2 Member Typedef Documentation

11.199.2.1 template<typename T > typedef T Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >::modulo_type

Definition at line 420 of file Interval_Restriction.defs.hh.

11.199.3 Constructor & Destructor Documentation

11.199.3.1 template<typename T > template<typename From > Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >::Simple_Restriction_Integer_Modulo (const From & *r*, const From & *d*) [inline]

Definition at line 417 of file Interval_Restriction.defs.hh.

11.199.4 Member Data Documentation

11.199.4.1 template<typename T > T Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >::divisor

Definition at line 422 of file Interval_Restriction.defs.hh.

11.199.4.2 template<typename T > T Parma_Polyhedra_Library::Simple_Restriction_Integer_-Modulo< T >::remainder

Definition at line 421 of file Interval_Restriction.defs.hh.

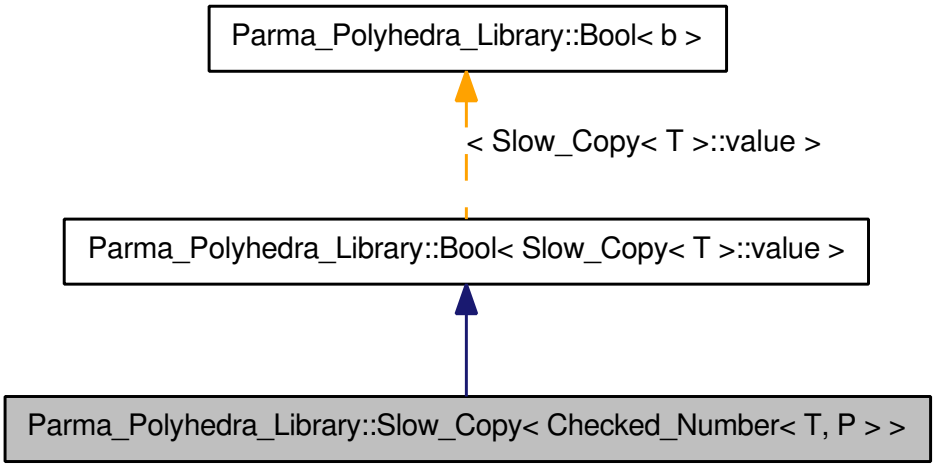
The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

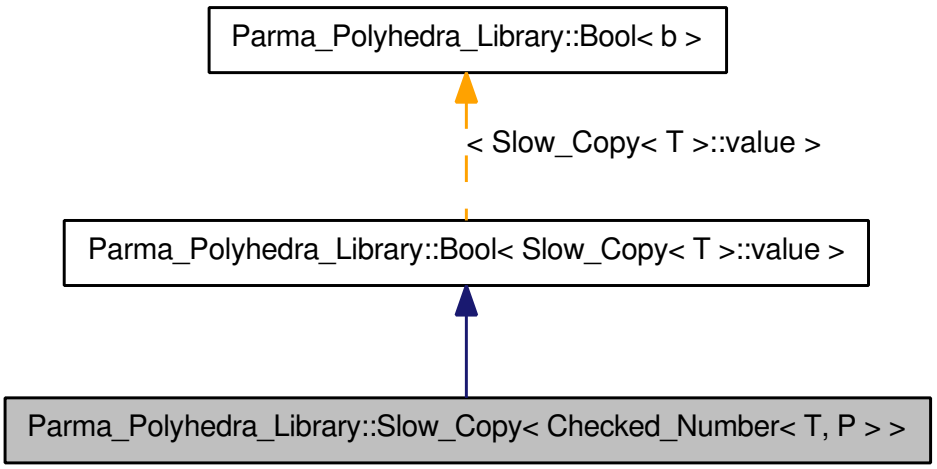
11.200 Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > > Struct Template Reference

`#include <Checked_Number.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >:



Collaboration diagram for Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >:



11.200.1 Detailed Description

template<typename T, typename P> struct Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >

Definition at line 431 of file Checked_Number.defs.hh.

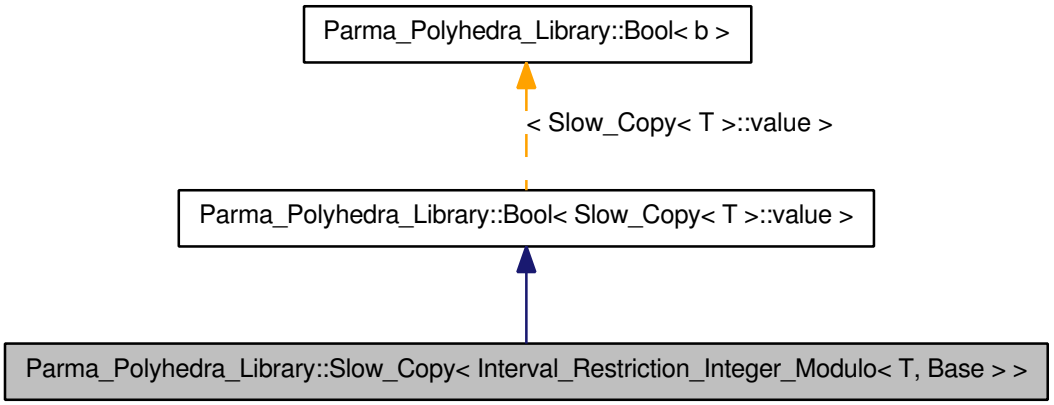
The documentation for this struct was generated from the following file:

- Checked_Number.defs.hh

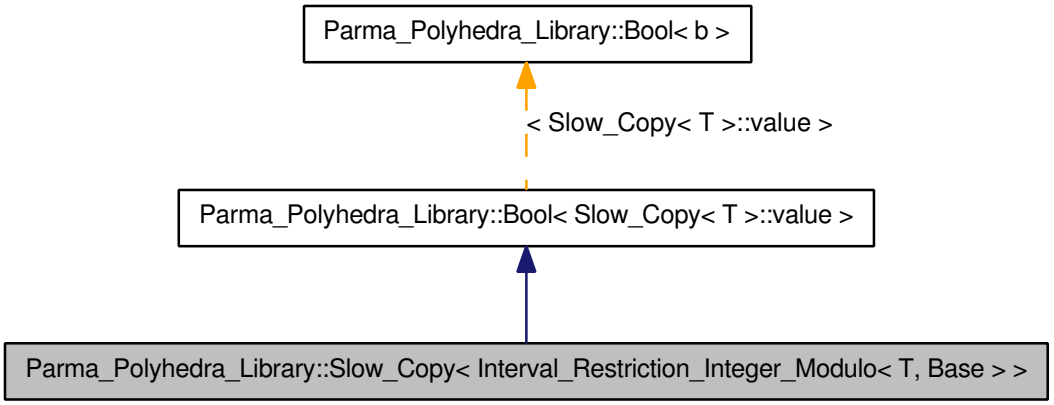
11.201 Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > > Struct Template Reference

`#include <Interval_Restriction.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > >:



Collaboration diagram for Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > >:



11.201.1 Detailed Description

template<typename T, typename Base> struct Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > >

Definition at line 403 of file Interval_Restriction.defs.hh.

The documentation for this struct was generated from the following file:

- Interval_Restriction.defs.hh

11.202 Parma_Polyhedra_Library::Smash_Reduction< D1, D2 > Class Template Reference

This class provides the reduction method for the Smash_Product domain.

`#include <Partially_Reduced_Product.defs.hh>`

Public Member Functions

- Smash_Reduction ()

  Default constructor.

- void product_reduce (D1 &d1, D2 &d2)

  The smash reduction operator for propagating emptiness between the domain elements `d1` and `d2`.

- ~Smash_Reduction ()

  Destructor.

11.202.1 Detailed Description

template<typename D1, typename D2> class Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >

This class provides the reduction method for the Smash_Product domain. The reduction classes are used to instantiate the Partially_Reduced_Product domain. This class propagates emptiness between its components.

Definition at line 101 of file Partially_Reduced_Product.defs.hh.

11.202.2 Constructor & Destructor Documentation

11.202.2.1 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >::Smash_Reduction () [inline]

Default constructor.

Definition at line 761 of file Partially_Reduced_Product.inlines.hh.

11.202.2.2 template<typename D1 , typename D2 > Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >::~Smash_Reduction () [inline]

Destructor.

Definition at line 766 of file Partially_Reduced_Product.inlines.hh.

11.202.3 Member Function Documentation

11.202.3.1 template<typename D1 , typename D2 > void Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >::product_reduce (D1 & *d1*, D2 & *d2*) [inline]

The smash reduction operator for propagating emptiness between the domain elements `d1` and `d2`.

If either of the the domain elements `d1` or `d2` is empty then the other is also set empty.

Parameters

d1 A pointset domain element;

d2 A pointset domain element;

Definition at line 454 of file Partially_Reduced_Product.templates.hh.

References Parma_Polyhedra_Library::EMPTY, and Parma_Polyhedra_Library::swap().

Referenced by Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >::product_reduce(), and Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >::product_reduce().

The documentation for this class was generated from the following files:

- Partially_Reduced_Product.defs.hh
- Partially_Reduced_Product.inlines.hh
- Partially_Reduced_Product.templates.hh

11.203 Parma_Polyhedra_Library::Grid::Status Class Reference

A conjunctive assertion about a grid.

Public Member Functions

- Status ()

  By default Status is the zero-dim universe assertion.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

Test, remove or add an individual assertion from the conjunction

- bool test_zero_dim_univ () const
- void reset_zero_dim_univ ()
- void set_zero_dim_univ ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_c_up_to_date () const
- void reset_c_up_to_date ()
- void set_c_up_to_date ()
- bool test_g_up_to_date () const
- void reset_g_up_to_date ()
- void set_g_up_to_date ()
- bool test_c_minimized () const
- void reset_c_minimized ()
- void set_c_minimized ()
- bool test_g_minimized () const
- void reset_g_minimized ()
- void set_g_minimized ()
- bool test_sat_c_up_to_date () const
- void reset_sat_c_up_to_date ()
- void set_sat_c_up_to_date ()
- bool test_sat_g_up_to_date () const
- void reset_sat_g_up_to_date ()
- void set_sat_g_up_to_date ()
- bool test_c_pending () const
- void reset_c_pending ()
- void set_c_pending ()
- bool test_g_pending () const
- void reset_g_pending ()
- void set_g_pending ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bitmask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` *are set.*

- bool test_any (flags_t mask) const

  Check whether at least one bit in `mask` *is set.*

- void set (flags_t mask)

  Set the bits in `mask`*.*

- void reset (flags_t mask)

  Reset the bits in `mask`*.*

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bitmasks for the individual assertions

- static const flags_t ZERO_DIM_UNIV = 0U
- static const flags_t EMPTY = 1U << 0
- static const flags_t C_UP_TO_DATE = 1U << 1
- static const flags_t G_UP_TO_DATE = 1U << 2
- static const flags_t C_MINIMIZED = 1U << 3
- static const flags_t G_MINIMIZED = 1U << 4
- static const flags_t SAT_C_UP_TO_DATE = 1U << 5
- static const flags_t SAT_G_UP_TO_DATE = 1U << 6
- static const flags_t CS_PENDING = 1U << 7
- static const flags_t GS_PENDING = 1U << 8

11.203.1 Detailed Description

A conjunctive assertion about a grid. The assertions supported that are in use are:

- *zero-dim universe*: the grid is the zero-dimension vector space $\mathbb{R}^0 = \{\cdot\}$;
- *empty*: the grid is the empty set;
- *congruences up-to-date*: the grid is correctly characterized by the attached system of congruences, modulo the processing of pending generators;
- *generators up-to-date*: the grid is correctly characterized by the attached system of generators, modulo the processing of pending congruences;

- *congruences minimized*: the non-pending part of the system of congruences attached to the grid is in minimal form;
- *generators minimized*: the non-pending part of the system of generators attached to the grid is in minimal form.

Other supported assertions are:

- *congruences pending*
- *generators pending*
- *congruences' saturation matrix up-to-date*
- *generators' saturation matrix up-to-date*.

Not all the conjunctions of these elementary assertions constitute a legal Status. In fact:

- *zero-dim universe* excludes any other assertion;
- *empty*: excludes any other assertion;
- *congruences pending* and *generators pending* are mutually exclusive;
- *congruences pending* implies both *congruences minimized* and *generators minimized*;
- *generators pending* implies both *congruences minimized* and *generators minimized*;
- *congruences minimized* implies *congruences up-to-date*;
- *generators minimized* implies *generators up-to-date*;
- *congruences' saturation matrix up-to-date* implies both *congruences up-to-date* and *generators up-to-date*;
- *generators' saturation matrix up-to-date* implies both *congruences up-to-date* and *generators up-to-date*.

Definition at line 72 of file Grid.defs.hh.

11.203.2 Member Typedef Documentation

11.203.2.1 typedef unsigned int Parma_Polyhedra_Library::Grid::Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 64 of file Grid.defs.hh.

11.203.3 Constructor & Destructor Documentation

11.203.3.1 Parma_Polyhedra_Library::Grid::Status::Status ()

By default Status is the *zero-dim universe* assertion.

11.203.3.2 Parma_Polyhedra_Library::Grid::Status::Status (flags_t *mask*) [private]

Construct from a bitmask.

11.203.4 Member Function Documentation

11.203.4.1 void Parma_Polyhedra_Library::Grid::Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

11.203.4.2 void Parma_Polyhedra_Library::Grid::Status::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Grid::ascii_dump().

11.203.4.3 bool Parma_Polyhedra_Library::Grid::Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

11.203.4.4 bool Parma_Polyhedra_Library::Grid::Status::OK () const

Checks if all the invariants are satisfied.

Referenced by Parma_Polyhedra_Library::Grid::OK().

11.203.4.5 void Parma_Polyhedra_Library::Grid::Status::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.203.4.6 void Parma_Polyhedra_Library::Grid::Status::reset (flags_t *mask*) [private]

Reset the bits in `mask`.

11.203.4.7 void Parma_Polyhedra_Library::Grid::Status::reset_c_minimized ()

Referenced by Parma_Polyhedra_Library::Grid::clear_congruences_minimized().

11.203.4.8 void Parma_Polyhedra_Library::Grid::Status::reset_c_pending ()

11.203.4.9 void Parma_Polyhedra_Library::Grid::Status::reset_c_up_to_date ()

Referenced by Parma_Polyhedra_Library::Grid::clear_congruences_up_to_date().

11.203.4.10 void Parma_Polyhedra_Library::Grid::Status::reset_empty ()

Referenced by Parma_Polyhedra_Library::Grid::clear_empty().

11.203.4.11 void Parma_Polyhedra_Library::Grid::Status::reset_g_minimized ()

Referenced by Parma_Polyhedra_Library::Grid::clear_generators_minimized().

11.203.4.12 void Parma_Polyhedra_Library::Grid::Status::reset_g_pending ()

11.203.4.13 void Parma_Polyhedra_Library::Grid::Status::reset_g_up_to_date ()

Referenced by Parma_Polyhedra_Library::Grid::clear_generators_up_to_date().

11.203.4.14 void Parma_Polyhedra_Library::Grid::Status::reset_sat_c_up_to_date ()

11.203.4.15 void Parma_Polyhedra_Library::Grid::Status::reset_sat_g_up_to_date ()

11.203.4.16 void Parma_Polyhedra_Library::Grid::Status::reset_zero_dim_univ ()

11.203.4.17 void Parma_Polyhedra_Library::Grid::Status::set (flags_t *mask*) [private]

Set the bits in `mask`.

11.203.4.18 void Parma_Polyhedra_Library::Grid::Status::set_c_minimized ()

Referenced by Parma_Polyhedra_Library::Grid::set_congruences_minimized().

11.203.4.19 void Parma_Polyhedra_Library::Grid::Status::set_c_pending ()

11.203.4.20 void Parma_Polyhedra_Library::Grid::Status::set_c_up_to_date ()

Referenced by Parma_Polyhedra_Library::Grid::set_congruences_up_to_date().

11.203.4.21 void Parma_Polyhedra_Library::Grid::Status::set_empty ()

Referenced by Parma_Polyhedra_Library::Grid::construct(), Parma_Polyhedra_Library::Grid::Grid(), and Parma_Polyhedra_Library::Grid::set_empty().

11.203.4.22 void Parma_Polyhedra_Library::Grid::Status::set_g_minimized ()

Referenced by Parma_Polyhedra_Library::Grid::set_generators_minimized().

11.203.4.23 void Parma_Polyhedra_Library::Grid::Status::set_g_pending ()

11.203.4.24 void Parma_Polyhedra_Library::Grid::Status::set_g_up_to_date ()

Referenced by Parma_Polyhedra_Library::Grid::set_generators_up_to_date().

11.203.4.25 void Parma_Polyhedra_Library::Grid::Status::set_sat_c_up_to_date ()

11.203.4.26 void Parma_Polyhedra_Library::Grid::Status::set_sat_g_up_to_date ()

11.203.4.27 void Parma_Polyhedra_Library::Grid::Status::set_zero_dim_univ ()

Referenced by Parma_Polyhedra_Library::Grid::set_zero_dim_univ().

11.203.4.28 bool Parma_Polyhedra_Library::Grid::Status::test_all (flags_t *mask*) const [private]

Check whether *all* bits in `mask` are set.

11.203.4.29 bool Parma_Polyhedra_Library::Grid::Status::test_any (flags_t *mask*) const [private]

Check whether *at least one* bit in `mask` is set.

11.203.4.30 bool Parma_Polyhedra_Library::Grid::Status::test_c_minimized () const

Referenced by Parma_Polyhedra_Library::Grid::congruences_are_minimized().

11.203.4.31 bool Parma_Polyhedra_Library::Grid::Status::test_c_pending () const

11.203.4.32 bool Parma_Polyhedra_Library::Grid::Status::test_c_up_to_date () const

Referenced by Parma_Polyhedra_Library::Grid::congruences_are_up_to_date().

11.203.4.33 bool Parma_Polyhedra_Library::Grid::Status::test_empty () const

Referenced by Parma_Polyhedra_Library::Grid::marked_empty().

11.203.4.34 bool Parma_Polyhedra_Library::Grid::Status::test_g_minimized () const

Referenced by Parma_Polyhedra_Library::Grid::generators_are_minimized().

11.203.4.35 bool Parma_Polyhedra_Library::Grid::Status::test_g_pending () const

11.203.4.36 bool Parma_Polyhedra_Library::Grid::Status::test_g_up_to_date () const

Referenced by Parma_Polyhedra_Library::Grid::generators_are_up_to_date().

11.203.4.37 bool Parma_Polyhedra_Library::Grid::Status::test_sat_c_up_to_date () const

11.203.4.38 bool Parma_Polyhedra_Library::Grid::Status::test_sat_g_up_to_date () const

11.203.4.39 bool Parma_Polyhedra_Library::Grid::Status::test_zero_dim_univ () const

Referenced by Parma_Polyhedra_Library::Grid::add_space_dimensions_and_embed(), and Parma_Polyhedra_Library::Grid::add_space_dimensions_and_project().

11.203.5 Member Data Documentation

11.203.5.1 const flags_t Parma_Polyhedra_Library::Grid::Status::C_MINIMIZED = 1U << 3 [static, private]

Definition at line 72 of file Grid.defs.hh.

11.203.5.2 const flags_t Parma_Polyhedra_Library::Grid::Status::C_UP_TO_DATE = 1U << 1 [static, private]

Definition at line 70 of file Grid.defs.hh.

11.203.5.3 const flags_t Parma_Polyhedra_Library::Grid::Status::CS_PENDING = 1U << 7 [static, private]

Definition at line 76 of file Grid.defs.hh.

11.203.5.4 const flags_t Parma_Polyhedra_Library::Grid::Status::EMPTY = 1U << 0 [static, private]

Definition at line 69 of file Grid.defs.hh.

11.203.5.5 flags_t Parma_Polyhedra_Library::Grid::Status::flags [private]

This holds the current bitset.

Definition at line 81 of file Grid.defs.hh.

11.203.5.6 const flags_t Parma_Polyhedra_Library::Grid::Status::G_MINIMIZED = 1U << 4 [static, private]

Definition at line 73 of file Grid.defs.hh.

11.203.5.7 const flags_t Parma_Polyhedra_Library::Grid::Status::G_UP_TO_DATE = 1U << 2 [static, private]

Definition at line 71 of file Grid.defs.hh.

11.203.5.8 const flags_t Parma_Polyhedra_Library::Grid::Status::GS_PENDING = 1U << 8 [static, private]

Definition at line 77 of file Grid.defs.hh.

11.203.5.9 const flags_t Parma_Polyhedra_Library::Grid::Status::SAT_C_UP_TO_DATE = 1U << 5 [static, private]

Definition at line 74 of file Grid.defs.hh.

11.203.5.10 const flags_t Parma_Polyhedra_Library::Grid::Status::SAT_G_UP_TO_DATE = 1U << 6 [static, private]

Definition at line 75 of file Grid.defs.hh.

11.203.5.11 const flags_t Parma_Polyhedra_Library::Grid::Status::ZERO_DIM_UNIV = 0U [static, private]

Definition at line 68 of file Grid.defs.hh.

The documentation for this class was generated from the following file:

- Grid.defs.hh

11.204 Parma_Polyhedra_Library::Box< ITV >::Status Class Reference

Public Member Functions

- Status ()

  By default Status is the empty set of assertion.

- Status (const Status &y)

  Ordinary copy constructor.

- template<typename Other_ITV >
  Status (const typename Box< Other_ITV >::Status &y)

  Copy constructor from a box of different type.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Test, remove or add an individual assertion from the conjunction.

- bool test_empty_up_to_date () const
- void reset_empty_up_to_date ()
- void set_empty_up_to_date ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_universe () const
- void reset_universe ()
- void set_universe ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bit-mask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` are set.

- bool test_any (flags_t mask) const

  Check whether at least one bit in `mask` is set.

- void set (flags_t mask)

  Set the bits in `mask`.

- void reset (flags_t mask)

  Reset the bits in `mask`.

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bit-masks for the individual assertions.

- static const flags_t NONE = 0U
- static const flags_t EMPTY_UP_TO_DATE = 1U << 0
- static const flags_t EMPTY = 1U << 1
- static const flags_t UNIVERSE = 1U << 2

11.204.1 Detailed Description

template<typename ITV> class Parma_Polyhedra_Library::Box< ITV >::Status

Definition at line 46 of file Box.defs.hh.

11.204.2 Member Typedef Documentation

11.204.2.1 template<typename ITV> typedef unsigned int Parma_Polyhedra_Library::Box< ITV >::Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 87 of file Box.defs.hh.

11.204.3 Constructor & Destructor Documentation

11.204.3.1 template<typename ITV> Parma_Polyhedra_Library::Box< ITV >::Status::Status ()

By default Status is the empty set of assertion.

11.204.3.2 template<typename ITV> Parma_Polyhedra_Library::Box< ITV >::Status::Status (const Status & *y*)

Ordinary copy constructor.

11.204.3.3 template<typename ITV> template<typename Other_ITV > Parma_Polyhedra_Library::Box< ITV >::Status::Status (const typename Box< Other_ITV >::Status & *y*) [inline]

Copy constructor from a box of different type.

11.204.3.4 template<typename ITV> Parma_Polyhedra_Library::Box< ITV >::Status::Status (flags_t *mask*) [private]

Construct from a bit-mask.

11.204.4 Member Function Documentation

11.204.4.1 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

11.204.4.2 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Box< ITV >::ascii_dump().

11.204.4.3 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

11.204.4.4 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::OK () const

Checks if all the invariants are satisfied.

11.204.4.5 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.204.4.6 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::reset (flags_t *mask*) [private]

Reset the bits in `mask`.

11.204.4.7 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::reset_empty ()

Referenced by Parma_Polyhedra_Library::Box< ITV >::set_nonempty().

11.204.4.8 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::reset_empty_up_to_date ()

Referenced by Parma_Polyhedra_Library::Box< ITV >::reset_empty_up_to_date().

11.204.4.9 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::reset_universe ()

11.204.4.10 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::set (flags_t *mask*) [private]

Set the bits in `mask`.

11.204.4.11 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::set_empty ()

Referenced by Parma_Polyhedra_Library::Box< ITV >::set_empty().

11.204.4.12 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::set_empty_up_to_date ()

Referenced by Parma_Polyhedra_Library::Box< ITV >::set_empty(), Parma_Polyhedra_Library::Box< ITV >::set_empty_up_to_date(), and Parma_Polyhedra_Library::Box< ITV >::set_nonempty().

11.204.4.13 template<typename ITV> void Parma_Polyhedra_Library::Box< ITV >::Status::set_universe ()

11.204.4.14 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::test_all (flags_t *mask*) const [private]

Check whether *all* bits in `mask` are set.

11.204.4.15 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::test_any (flags_t *mask*) const [private]

Check whether *at least one* bit in `mask` is set.

11.204.4.16 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::test_empty () const

Referenced by Parma_Polyhedra_Library::Box< ITV >::marked_empty(), and Parma_Polyhedra_Library::Box< ITV >::OK().

11.204.4.17 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::test_empty_up_to_date () const

Referenced by Parma_Polyhedra_Library::Box< ITV >::concatenate_assign(), Parma_Polyhedra_Library::Box< ITV >::marked_empty(), and Parma_Polyhedra_Library::Box< ITV >::OK().

11.204.4.18 template<typename ITV> bool Parma_Polyhedra_Library::Box< ITV >::Status::test_universe () const

11.204.5 Member Data Documentation

11.204.5.1 template<typename ITV> const flags_t Parma_Polyhedra_Library::Box< ITV >::Status::EMPTY = 1U << 1 [static, private]

Definition at line 93 of file Box.defs.hh.

11.204.5.2 template<typename ITV> const flags_t Parma_Polyhedra_Library::Box< ITV >::Status::EMPTY_UP_TO_DATE = 1U << 0 [static, private]

Definition at line 92 of file Box.defs.hh.

11.204.5.3 template<typename ITV> flags_t Parma_Polyhedra_Library::Box< ITV >::Status::flags [private]

This holds the current bitset.

Definition at line 98 of file Box.defs.hh.

11.204.5.4 template<typename ITV> const flags_t Parma_Polyhedra_Library::Box< ITV >::Status::NONE = 0U [static, private]

Definition at line 91 of file Box.defs.hh.

11.204.5.5 template<typename ITV> const flags_t Parma_Polyhedra_Library::Box< ITV >::Status::UNIVERSE = 1U << 2 [static, private]

Definition at line 94 of file Box.defs.hh.

The documentation for this class was generated from the following file:

- Box.defs.hh

11.205 Parma_Polyhedra_Library::BD_Shape< T >::Status Class Reference

A conjunctive assertion about a BD_Shape<T> object.

Public Member Functions

- Status ()

  By default Status is the zero-dim universe assertion.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Test, remove or add an individual assertion from the conjunction.

- bool test_zero_dim_univ () const
- void reset_zero_dim_univ ()
- void set_zero_dim_univ ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_shortest_path_closed () const
- void reset_shortest_path_closed ()
- void set_shortest_path_closed ()
- bool test_shortest_path_reduced () const
- void reset_shortest_path_reduced ()
- void set_shortest_path_reduced ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bit-mask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` are set.

- bool test_any (flags_t mask) const

Check whether at least one bit in `mask` *is set.*

- void set (flags_t mask)

  Set the bits in `mask`*.*

- void reset (flags_t mask)

  Reset the bits in `mask`*.*

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bit-masks for the individual assertions.

- static const flags_t ZERO_DIM_UNIV = 0U
- static const flags_t EMPTY = 1U << 0
- static const flags_t SHORTEST_PATH_CLOSED = 1U << 1
- static const flags_t SHORTEST_PATH_REDUCED = 1U << 2

11.205.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::BD_Shape< T >::Status

A conjunctive assertion about a BD_Shape<T> object. The assertions supported are:

- *zero-dim universe*: the BDS is the zero-dimensional vector space $\mathbb{R}^0 = \{\cdot\}$;
- *empty*: the BDS is the empty set;
- *shortest-path closed*: the BDS is represented by a shortest-path closed system of bounded differences, so that all the constraints are as tight as possible;
- *shortest-path reduced*: the BDS is represented by a shortest-path closed system of bounded differences and each constraint in such a system is marked as being either redundant or non-redundant.

Not all the conjunctions of these elementary assertions constitute a legal Status. In fact:

- *zero-dim universe* excludes any other assertion;
- *empty*: excludes any other assertion;
- *shortest-path reduced* implies *shortest-path closed*.

Definition at line 51 of file BD_Shape.defs.hh.

11.205.2 Member Typedef Documentation

11.205.2.1 template<typename T> typedef unsigned int Parma_Polyhedra_Library::BD_Shape< T >::Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 40 of file BD_Shape.defs.hh.

11.205.3 Constructor & Destructor Documentation

11.205.3.1 template<typename T> Parma_Polyhedra_Library::BD_Shape< T >::Status::Status ()

By default Status is the *zero-dim universe* assertion.

11.205.3.2 template<typename T> Parma_Polyhedra_Library::BD_Shape< T >::Status::Status (flags_t *mask*) [private]

Construct from a bit-mask.

11.205.4 Member Function Documentation

11.205.4.1 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

11.205.4.2 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::ascii_dump().

11.205.4.3 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

11.205.4.4 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::OK () const

Checks if all the invariants are satisfied.

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::OK().

11.205.4.5 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.205.4.6 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::reset (flags_t *mask*) [private]

Reset the bits in `mask`.

11.205.4.7 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_empty ()

11.205.4.8 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_shortest_path_closed ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_closed().

11.205.4.9 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_shortest_path_reduced ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::reset_shortest_path_reduced().

11.205.4.10 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::reset_zero_dim_univ ()

11.205.4.11 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::set (flags_t *mask*) [private]

Set the bits in `mask`.

11.205.4.12 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::set_empty ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::set_empty().

11.205.4.13 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::set_shortest_path_closed ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_closed().

11.205.4.14 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::set_shortest_path_reduced ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::set_shortest_path_reduced().

11.205.4.15 template<typename T> void Parma_Polyhedra_Library::BD_Shape< T >::Status::set_zero_dim_univ ()

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::set_zero_dim_univ().

11.205.4.16 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_all (flags_t *mask*) const [private]

Check whether *all* bits in `mask` are set.

11.205.4.17 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_any (flags_t *mask*) const [private]

Check whether *at least one* bit in `mask` is set.

11.205.4.18 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_empty () const

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::marked_empty().

11.205.4.19 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_shortest_path_closed () const

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_closed().

11.205.4.20 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_shortest_path_reduced () const

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::marked_shortest_path_reduced().

11.205.4.21 template<typename T> bool Parma_Polyhedra_Library::BD_Shape< T >::Status::test_zero_dim_univ () const

Referenced by Parma_Polyhedra_Library::BD_Shape< T >::marked_zero_dim_univ().

11.205.5 Member Data Documentation

11.205.5.1 template<typename T> const flags_t Parma_Polyhedra_Library::BD_Shape< T >::Status::EMPTY = 1U << 0 [static, private]

Definition at line 45 of file BD_Shape.defs.hh.

11.205.5.2 template<typename T> flags_t Parma_Polyhedra_Library::BD_Shape< T >::Status::flags [private]

This holds the current bitset.

Definition at line 51 of file BD_Shape.defs.hh.

11.205.5.3 template<typename T> const flags_t Parma_Polyhedra_Library::BD_Shape< T >::Status::SHORTEST_PATH_CLOSED = 1U << 1 [static, private]

Definition at line 46 of file BD_Shape.defs.hh.

11.205.5.4 template<typename T> const flags_t Parma_Polyhedra_Library::BD_Shape< T >::Status::SHORTEST_PATH_REDUCED = 1U << 2 [static, private]

Definition at line 47 of file BD_Shape.defs.hh.

11.205.5.5 template<typename T> const flags_t Parma_Polyhedra_Library::BD_Shape< T >::Status::ZERO_DIM_UNIV = 0U [static, private]

Definition at line 44 of file BD_Shape.defs.hh.

The documentation for this class was generated from the following file:

- BD_Shape.defs.hh

11.206 Status Class Reference

A conjunctive assertion about a polyhedron.

`#include <Ph_Status.idefs.hh>`

Public Member Functions

- Status ()

  By default Status is the zero-dim universe assertion.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` an ASCII representation of `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of `*this`.

- void print () const

  Prints `*this` to `std::cerr` using `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Test, remove or add an individual assertion from the conjunction

- bool test_zero_dim_univ () const
- void reset_zero_dim_univ ()
- void set_zero_dim_univ ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_c_up_to_date () const
- void reset_c_up_to_date ()
- void set_c_up_to_date ()
- bool test_g_up_to_date () const
- void reset_g_up_to_date ()
- void set_g_up_to_date ()
- bool test_c_minimized () const
- void reset_c_minimized ()
- void set_c_minimized ()
- bool test_g_minimized () const
- void reset_g_minimized ()

- void set_g_minimized ()
- bool test_sat_c_up_to_date () const
- void reset_sat_c_up_to_date ()
- void set_sat_c_up_to_date ()
- bool test_sat_g_up_to_date () const
- void reset_sat_g_up_to_date ()
- void set_sat_g_up_to_date ()
- bool test_c_pending () const
- void reset_c_pending ()
- void set_c_pending ()
- bool test_g_pending () const
- void reset_g_pending ()
- void set_g_pending ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bit-mask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` are set.

- bool test_any (flags_t mask) const

  Check whether at least one bit in `mask` is set.

- void set (flags_t mask)

  Set the bits in `mask`.

- void reset (flags_t mask)

  Reset the bits in `mask`.

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bit-masks for the individual assertions

- static const flags_t ZERO_DIM_UNIV = 0U
- static const flags_t EMPTY = 1U << 0
- static const flags_t C_UP_TO_DATE = 1U << 1
- static const flags_t G_UP_TO_DATE = 1U << 2

- static const flags_t C_MINIMIZED = 1U << 3
- static const flags_t G_MINIMIZED = 1U << 4
- static const flags_t SAT_C_UP_TO_DATE = 1U << 5
- static const flags_t SAT_G_UP_TO_DATE = 1U << 6
- static const flags_t CS_PENDING = 1U << 7
- static const flags_t GS_PENDING = 1U << 8

11.206.1 Detailed Description

A conjunctive assertion about a polyhedron. The assertions supported are:

- *zero-dim universe*: the polyhedron is the zero-dimension vector space $\mathbb{R}^0 = \{\cdot\}$;
- *empty*: the polyhedron is the empty set;
- *constraints pending*: the polyhedron is correctly characterized by the attached system of constraints, which is split in two non-empty subsets: the already processed constraints, which are in minimal form, and the pending constraints, which still have to be processed and may thus be inconsistent or contain redundancies;
- *generators pending*: the polyhedron is correctly characterized by the attached system of generators, which is split in two non-empty subsets: the already processed generators, which are in minimal form, and the pending generators, which still have to be processed and may thus contain redundancies;
- *constraints up-to-date*: the polyhedron is correctly characterized by the attached system of constraints, modulo the processing of pending generators;
- *generators up-to-date*: the polyhedron is correctly characterized by the attached system of generators, modulo the processing of pending constraints;
- *constraints minimized*: the non-pending part of the system of constraints attached to the polyhedron is in minimal form;
- *generators minimized*: the non-pending part of the system of generators attached to the polyhedron is in minimal form;
- *constraints' saturation matrix up-to-date*: the attached saturation matrix having rows indexed by non-pending generators and columns indexed by non-pending constraints correctly expresses the saturation relation between the attached non-pending constraints and generators;
- *generators' saturation matrix up-to-date*: the attached saturation matrix having rows indexed by non-pending constraints and columns indexed by non-pending generators correctly expresses the saturation relation between the attached non-pending constraints and generators;

Not all the conjunctions of these elementary assertions constitute a legal Status. In fact:

- *zero-dim universe* excludes any other assertion;
- *empty*: excludes any other assertion;
- *constraints pending* and *generators pending* are mutually exclusive;
- *constraints pending* implies both *constraints minimized* and *generators minimized*;
- *generators pending* implies both *constraints minimized* and *generators minimized*;
- *constraints minimized* implies *constraints up-to-date*;

- *generators minimized* implies *generators up-to-date*;
- *constraints' saturation matrix up-to-date* implies both *constraints up-to-date* and *generators up-to-date*;
- *generators' saturation matrix up-to-date* implies both *constraints up-to-date* and *generators up-to-date*.

Definition at line 86 of file Ph_Status.idefs.hh.

11.206.2 Member Typedef Documentation

11.206.2.1 typedef unsigned int Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 148 of file Ph_Status.idefs.hh.

11.206.3 Constructor & Destructor Documentation

11.206.3.1 Status::Status ()

By default Status is the *zero-dim universe* assertion.

11.206.3.2 Status::Status (flags_t *mask*) [private]

Construct from a bit-mask.

11.206.4 Member Function Documentation

11.206.4.1 void Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

11.206.4.2 void Status::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

11.206.4.3 bool Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

11.206.4.4 bool Status::OK () const

Checks if all the invariants are satisfied.

11.206.4.5 void Status::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.206.4.6 void Status::reset (flags_t *mask*) [private]

Reset the bits in `mask`.

11.206.4.7 void Status::reset_c_minimized ()

11.206.4.8 void Status::reset_c_pending ()

11.206.4.9 void Status::reset_c_up_to_date ()

11.206.4.10 void Status::reset_empty ()

11.206.4.11 void Status::reset_g_minimized ()

11.206.4.12 void Status::reset_g_pending ()

11.206.4.13 void Status::reset_g_up_to_date ()

11.206.4.14 void Status::reset_sat_c_up_to_date ()

11.206.4.15 void Status::reset_sat_g_up_to_date ()

11.206.4.16 void Status::reset_zero_dim_univ ()

11.206.4.17 void Status::set (flags_t *mask*) [private]

Set the bits in `mask`.

11.206.4.18 void Status::set_c_minimized ()

11.206.4.19 void Status::set_c_pending ()

11.206.4.20 void Status::set_c_up_to_date ()

11.206.4.21 void Status::set_empty ()

11.206.4.22 void Status::set_g_minimized ()

11.206.4.23 void Status::set_g_pending ()

11.206.4.24 void Status::set_g_up_to_date ()

11.206.4.25 void Status::set_sat_c_up_to_date ()

11.206.4.26 void Status::set_sat_g_up_to_date ()

11.206.4.27 void Status::set_zero_dim_univ ()

11.206.4.28 bool Status::test_all (flags_t *mask*) const [private]

Check whether *all* bits in `mask` are set.

11.206.4.29 bool Status::test_any (flags_t *mask*) const [private]

Check whether *at least one* bit in `mask` is set.

11.206.4.30 bool Status::test_c_minimized () const

11.206.4.31 bool Status::test_c_pending () const

11.206.4.32 bool Status::test_c_up_to_date () const

11.206.4.33 bool Status::test_empty () const

11.206.4.34 bool Status::test_g_minimized () const

11.206.4.35 bool Status::test_g_pending () const

11.206.4.36 bool Status::test_g_up_to_date () const

11.206.4.37 bool Status::test_sat_c_up_to_date () const

11.206.4.38 bool Status::test_sat_g_up_to_date () const

11.206.4.39 bool Status::test_zero_dim_univ () const

11.206.5 Member Data Documentation

11.206.5.1 const flags_t Status::C_MINIMIZED = 1U << 3 [static, private]

Definition at line 156 of file Ph_Status.idefs.hh.

11.206.5.2 const flags_t Status::C_UP_TO_DATE = 1U << 1 [static, private]

Definition at line 154 of file Ph_Status.idefs.hh.

11.206.5.3 const flags_t Status::CS_PENDING = 1U << 7 [static, private]

Definition at line 160 of file Ph_Status.idefs.hh.

11.206.5.4 const flags_t Status::EMPTY = 1U << 0 [static, private]

Definition at line 153 of file Ph_Status.idefs.hh.

11.206.5.5 flags_t Status::flags [private]

This holds the current bitset.

Definition at line 165 of file Ph_Status.idefs.hh.

11.206.5.6 const flags_t Status::G_MINIMIZED = 1U << 4 [static, private]

Definition at line 157 of file Ph_Status.idefs.hh.

11.206.5.7 const flags_t Status::G_UP_TO_DATE = 1U << 2 [static, private]

Definition at line 155 of file Ph_Status.idefs.hh.

11.206.5.8 const flags_t Status::GS_PENDING = 1U << 8 [static, private]

Definition at line 161 of file Ph_Status.idefs.hh.

11.206.5.9 const flags_t Status::SAT_C_UP_TO_DATE = 1U << 5 [static, private]

Definition at line 158 of file Ph_Status.idefs.hh.

11.206.5.10 const flags_t Status::SAT_G_UP_TO_DATE = 1U << 6 [static, private]

Definition at line 159 of file Ph_Status.idefs.hh.

11.206.5.11 const flags_t Status::ZERO_DIM_UNIV = 0U [static, private]

Definition at line 152 of file Ph_Status.idefs.hh.

The documentation for this class was generated from the following file:

- Ph_Status.idefs.hh

11.207 Parma_Polyhedra_Library::Polyhedron::Status Class Reference

A conjunctive assertion about a polyhedron.

Public Member Functions

- Status ()

  By default Status is the zero-dim universe assertion.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

- bool ascii_load (std::istream &s)

  Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

Test, remove or add an individual assertion from the conjunction

- bool test_zero_dim_univ () const
- void reset_zero_dim_univ ()
- void set_zero_dim_univ ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_c_up_to_date () const
- void reset_c_up_to_date ()
- void set_c_up_to_date ()
- bool test_g_up_to_date () const
- void reset_g_up_to_date ()
- void set_g_up_to_date ()
- bool test_c_minimized () const
- void reset_c_minimized ()
- void set_c_minimized ()
- bool test_g_minimized () const
- void reset_g_minimized ()
- void set_g_minimized ()
- bool test_sat_c_up_to_date () const
- void reset_sat_c_up_to_date ()
- void set_sat_c_up_to_date ()
- bool test_sat_g_up_to_date () const
- void reset_sat_g_up_to_date ()
- void set_sat_g_up_to_date ()
- bool test_c_pending () const
- void reset_c_pending ()
- void set_c_pending ()
- bool test_g_pending () const
- void reset_g_pending ()
- void set_g_pending ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bit-mask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` *are set.*

- bool test_any (flags_t mask) const

  Check whether at least one bit in `mask` *is set.*

- void set (flags_t mask)

  Set the bits in `mask`*.*

- void reset (flags_t mask)

  Reset the bits in `mask`*.*

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bit-masks for the individual assertions

- static const flags_t ZERO_DIM_UNIV = 0U
- static const flags_t EMPTY = 1U << 0
- static const flags_t C_UP_TO_DATE = 1U << 1
- static const flags_t G_UP_TO_DATE = 1U << 2
- static const flags_t C_MINIMIZED = 1U << 3
- static const flags_t G_MINIMIZED = 1U << 4
- static const flags_t SAT_C_UP_TO_DATE = 1U << 5
- static const flags_t SAT_G_UP_TO_DATE = 1U << 6
- static const flags_t CS_PENDING = 1U << 7
- static const flags_t GS_PENDING = 1U << 8

Related Functions

(Note that these are not member functions.)

- bool get_field (std::istream &s, const char *keyword, bool &positive)

11.207.1 Detailed Description

A conjunctive assertion about a polyhedron. The assertions supported are:

- *zero-dim universe*: the polyhedron is the zero-dimension vector space $\mathbb{R}^0 = \{\cdot\}$;

- *empty*: the polyhedron is the empty set;
- *constraints pending*: the polyhedron is correctly characterized by the attached system of constraints, which is split in two non-empty subsets: the already processed constraints, which are in minimal form, and the pending constraints, which still have to be processed and may thus be inconsistent or contain redundancies;
- *generators pending*: the polyhedron is correctly characterized by the attached system of generators, which is split in two non-empty subsets: the already processed generators, which are in minimal form, and the pending generators, which still have to be processed and may thus contain redundancies;
- *constraints up-to-date*: the polyhedron is correctly characterized by the attached system of constraints, modulo the processing of pending generators;
- *generators up-to-date*: the polyhedron is correctly characterized by the attached system of generators, modulo the processing of pending constraints;
- *constraints minimized*: the non-pending part of the system of constraints attached to the polyhedron is in minimal form;
- *generators minimized*: the non-pending part of the system of generators attached to the polyhedron is in minimal form;
- *constraints' saturation matrix up-to-date*: the attached saturation matrix having rows indexed by non-pending generators and columns indexed by non-pending constraints correctly expresses the saturation relation between the attached non-pending constraints and generators;
- *generators' saturation matrix up-to-date*: the attached saturation matrix having rows indexed by non-pending constraints and columns indexed by non-pending generators correctly expresses the saturation relation between the attached non-pending constraints and generators;

Not all the conjunctions of these elementary assertions constitute a legal Status. In fact:

- *zero-dim universe* excludes any other assertion;
- *empty*: excludes any other assertion;
- *constraints pending* and *generators pending* are mutually exclusive;
- *constraints pending* implies both *constraints minimized* and *generators minimized*;
- *generators pending* implies both *constraints minimized* and *generators minimized*;
- *constraints minimized* implies *constraints up-to-date*;
- *generators minimized* implies *generators up-to-date*;
- *constraints' saturation matrix up-to-date* implies both *constraints up-to-date* and *generators up-to-date*;
- *generators' saturation matrix up-to-date* implies both *constraints up-to-date* and *generators up-to-date*.

Definition at line 87 of file Polyhedron.defs.hh.

11.207.2 Member Typedef Documentation

11.207.2.1 typedef unsigned int Parma_Polyhedra_Library::Polyhedron::Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 64 of file Polyhedron.defs.hh.

11.207.3 Constructor & Destructor Documentation

11.207.3.1 Parma_Polyhedra_Library::Polyhedron::Status::Status () [inline]

By default Status is the *zero-dim universe* assertion.

Definition at line 35 of file Ph_Status.inlines.hh.

11.207.3.2 Parma_Polyhedra_Library::Polyhedron::Status::Status (flags_t *mask*) [inline, private]

Construct from a bit-mask.

Definition at line 30 of file Ph_Status.inlines.hh.

11.207.4 Member Function Documentation

11.207.4.1 void Parma_Polyhedra_Library::Polyhedron::Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 68 of file Ph_Status.cc.

References test_c_minimized(), test_c_pending(), test_c_up_to_date(), test_empty(), test_g_minimized(), test_g_pending(), test_g_up_to_date(), test_sat_c_up_to_date(), test_sat_g_up_to_date(), and test_zero_-dim_univ().

11.207.4.2 void Parma_Polyhedra_Library::Polyhedron::Status::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::Polyhedron::ascii_dump().

11.207.4.3 bool Parma_Polyhedra_Library::Polyhedron::Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 88 of file Ph_Status.cc.

References Parma_Polyhedra_Library::Polyhedron::OK(), Parma_Polyhedra_Library::Polyhedron::set_empty(), Parma_Polyhedra_Library::Polyhedron::set_sat_c_up_to_date(), Parma_Polyhedra_Library::Polyhedron::set_sat_g_up_to_date(), and Parma_Polyhedra_Library::Polyhedron::set_zero_dim_univ().

11.207.4.4 bool Parma_Polyhedra_Library::Polyhedron::Status::OK () const

Checks if all the invariants are satisfied.

Definition at line 163 of file Ph_Status.cc.

References reset_empty(), test_c_minimized(), test_c_pending(), test_c_up_to_date(), test_empty(), test_g_minimized(), test_g_pending(), test_g_up_to_date(), test_sat_c_up_to_date(), test_sat_g_up_to_date(), and test_zero_dim_univ().

Referenced by Parma_Polyhedra_Library::Polyhedron::OK().

11.207.4.5 void Parma_Polyhedra_Library::Polyhedron::Status::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.207.4.6 void Parma_Polyhedra_Library::Polyhedron::Status::reset (flags_t *mask*) [inline, private]

Reset the bits in `mask`.

Definition at line 55 of file Ph_Status.inlines.hh.

References flags.

Referenced by reset_c_minimized(), reset_c_pending(), reset_c_up_to_date(), reset_empty(), reset_g_minimized(), reset_g_pending(), reset_g_up_to_date(), reset_sat_c_up_to_date(), and reset_sat_g_up_to_date().

11.207.4.7 void Parma_Polyhedra_Library::Polyhedron::Status::reset_c_minimized () [inline]

Definition at line 129 of file Ph_Status.inlines.hh.

References C_MINIMIZED, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_constraints_minimized().

11.207.4.8 void Parma_Polyhedra_Library::Polyhedron::Status::reset_c_pending () [inline]

Definition at line 160 of file Ph_Status.inlines.hh.

References CS_PENDING, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_pending_constraints().

11.207.4.9 void Parma_Polyhedra_Library::Polyhedron::Status::reset_c_up_to_date () [inline]

Definition at line 99 of file Ph_Status.inlines.hh.

References C_UP_TO_DATE, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_constraints_up_to_date().

11.207.4.10 void Parma_Polyhedra_Library::Polyhedron::Status::reset_empty () [inline]

Definition at line 84 of file Ph_Status.inlines.hh.

References EMPTY, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_empty(), and OK().

11.207.4.11 void Parma_Polyhedra_Library::Polyhedron::Status::reset_g_minimized () [inline]

Definition at line 144 of file Ph_Status.inlines.hh.

References G_MINIMIZED, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_generators_minimized().

11.207.4.12 void Parma_Polyhedra_Library::Polyhedron::Status::reset_g_pending () [inline]

Definition at line 175 of file Ph_Status.inlines.hh.

References GS_PENDING, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_pending_generators().

11.207.4.13 void Parma_Polyhedra_Library::Polyhedron::Status::reset_g_up_to_date () [inline]

Definition at line 114 of file Ph_Status.inlines.hh.

References G_UP_TO_DATE, and reset().

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_generators_up_to_date().

11.207.4.14 void Parma_Polyhedra_Library::Polyhedron::Status::reset_sat_c_up_to_date () [inline]

Definition at line 191 of file Ph_Status.inlines.hh.

References reset(), and SAT_C_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_sat_c_up_to_date().

11.207.4.15 void Parma_Polyhedra_Library::Polyhedron::Status::reset_sat_g_up_to_date () [inline]

Definition at line 206 of file Ph_Status.inlines.hh.

References reset(), and SAT_G_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::clear_sat_g_up_to_date().

11.207.4.16 void Parma_Polyhedra_Library::Polyhedron::Status::reset_zero_dim_univ () [inline]

Definition at line 65 of file Ph_Status.inlines.hh.

References EMPTY, flags, and ZERO_DIM_UNIV.

11.207.4.17 void Parma_Polyhedra_Library::Polyhedron::Status::set (flags_t *mask*) [inline, private]

Set the bits in `mask`.

Definition at line 50 of file Ph_Status.inlines.hh.

References flags.

11.207.4.18 void Parma_Polyhedra_Library::Polyhedron::Status::set_c_minimized () [inline]

Definition at line 134 of file Ph_Status.inlines.hh.

References C_MINIMIZED.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_constraints_minimized().

11.207.4.19 void Parma_Polyhedra_Library::Polyhedron::Status::set_c_pending () [inline]

Definition at line 165 of file Ph_Status.inlines.hh.

References CS_PENDING.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_constraints_pending().

11.207.4.20 void Parma_Polyhedra_Library::Polyhedron::Status::set_c_up_to_date () [inline]

Definition at line 104 of file Ph_Status.inlines.hh.

References C_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_constraints_up_to_date().

11.207.4.21 void Parma_Polyhedra_Library::Polyhedron::Status::set_empty () [inline]

Definition at line 89 of file Ph_Status.inlines.hh.

References EMPTY, and flags.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_recycled_constraints(), Parma_Polyhedra_-Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Polyhedron::refine_with_constraints(), and Parma_Polyhedra_Library::Polyhedron::set_empty().

11.207.4.22 void Parma_Polyhedra_Library::Polyhedron::Status::set_g_minimized () [inline]

Definition at line 149 of file Ph_Status.inlines.hh.

References G_MINIMIZED.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_generators_minimized().

11.207.4.23 void Parma_Polyhedra_Library::Polyhedron::Status::set_g_pending () [inline]

Definition at line 180 of file Ph_Status.inlines.hh.

References GS_PENDING.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_generators_pending().

11.207.4.24 void Parma_Polyhedra_Library::Polyhedron::Status::set_g_up_to_date () [inline]

Definition at line 119 of file Ph_Status.inlines.hh.

References G_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_generators_up_to_date().

11.207.4.25 void Parma_Polyhedra_Library::Polyhedron::Status::set_sat_c_up_to_date () [inline]

Definition at line 196 of file Ph_Status.inlines.hh.

References SAT_C_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_sat_c_up_to_date().

11.207.4.26 void Parma_Polyhedra_Library::Polyhedron::Status::set_sat_g_up_to_date () [inline]

Definition at line 211 of file Ph_Status.inlines.hh.

References SAT_G_UP_TO_DATE.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_sat_g_up_to_date().

11.207.4.27 void Parma_Polyhedra_Library::Polyhedron::Status::set_zero_dim_univ () [inline]

Definition at line 73 of file Ph_Status.inlines.hh.

References flags, and ZERO_DIM_UNIV.

Referenced by Parma_Polyhedra_Library::Polyhedron::set_zero_dim_univ().

11.207.4.28 bool Parma_Polyhedra_Library::Polyhedron::Status::test_all (flags_t *mask*) const [inline, private]

Check whether *all* bits in `mask` are set.

Definition at line 40 of file Ph_Status.inlines.hh.

References flags.

11.207.4.29 bool Parma_Polyhedra_Library::Polyhedron::Status::test_any (flags_t *mask*) const [inline, private]

Check whether *at least one* bit in `mask` is set.

Definition at line 45 of file Ph_Status.inlines.hh.

References flags.

Referenced by test_c_minimized(), test_c_pending(), test_c_up_to_date(), test_empty(), test_g_-minimized(), test_g_pending(), test_g_up_to_date(), test_sat_c_up_to_date(), and test_sat_g_up_to_-date().

11.207.4.30 bool Parma_Polyhedra_Library::Polyhedron::Status::test_c_minimized () const [inline]

Definition at line 124 of file Ph_Status.inlines.hh.

References C_MINIMIZED, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::constraints_are_minimized(), and OK().

11.207.4.31 bool Parma_Polyhedra_Library::Polyhedron::Status::test_c_pending () const [inline]

Definition at line 155 of file Ph_Status.inlines.hh.

References CS_PENDING, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::has_pending_constraints(), Parma_-Polyhedra_Library::Polyhedron::has_something_pending(), and OK().

11.207.4.32 bool Parma_Polyhedra_Library::Polyhedron::Status::test_c_up_to_date () const [inline]

Definition at line 94 of file Ph_Status.inlines.hh.

References C_UP_TO_DATE, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::constraints_are_up_to_date(), and OK().

11.207.4.33 bool Parma_Polyhedra_Library::Polyhedron::Status::test_empty () const [inline]

Definition at line 79 of file Ph_Status.inlines.hh.

References EMPTY, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::marked_empty(), and OK().

11.207.4.34 bool Parma_Polyhedra_Library::Polyhedron::Status::test_g_minimized () const [inline]

Definition at line 139 of file Ph_Status.inlines.hh.

References G_MINIMIZED, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::generators_are_minimized(), and OK().

11.207.4.35 bool Parma_Polyhedra_Library::Polyhedron::Status::test_g_pending () const [inline]

Definition at line 170 of file Ph_Status.inlines.hh.

References GS_PENDING, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::has_pending_generators(), Parma_-Polyhedra_Library::Polyhedron::has_something_pending(), and OK().

11.207.4.36 bool Parma_Polyhedra_Library::Polyhedron::Status::test_g_up_to_date () const [inline]

Definition at line 109 of file Ph_Status.inlines.hh.

References G_UP_TO_DATE, and test_any().

Referenced by ascii_dump(), Parma_Polyhedra_Library::Polyhedron::generators_are_up_to_date(), and OK().

11.207.4.37 bool Parma_Polyhedra_Library::Polyhedron::Status::test_sat_c_up_to_date () const [inline]

Definition at line 186 of file Ph_Status.inlines.hh.

References SAT_C_UP_TO_DATE, and test_any().

Referenced by ascii_dump(), OK(), and Parma_Polyhedra_Library::Polyhedron::sat_c_is_up_to_date().

11.207.4.38 bool Parma_Polyhedra_Library::Polyhedron::Status::test_sat_g_up_to_date () const [inline]

Definition at line 201 of file Ph_Status.inlines.hh.

References SAT_G_UP_TO_DATE, and test_any().

Referenced by ascii_dump(), OK(), and Parma_Polyhedra_Library::Polyhedron::sat_g_is_up_to_date().

11.207.4.39 bool Parma_Polyhedra_Library::Polyhedron::Status::test_zero_dim_univ () const [inline]

Definition at line 60 of file Ph_Status.inlines.hh.

References flags, and ZERO_DIM_UNIV.

Referenced by Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::Polyhedron::add_space_dimensions_and_project(), ascii_dump(), and OK().

11.207.5 Friends And Related Function Documentation

11.207.5.1 bool get_field (std::istream & *s*, const char * *keyword*, bool & *positive*) [related]

Reads a keyword and its associated on/off, +/- flag from `s`. Returns `true` if the operation is successful, returns `false` otherwise. When successful, `positive` is set to `true` if the flag is on; it is set to `false` otherwise.

Definition at line 55 of file Ph_Status.cc.

11.207.6 Member Data Documentation

11.207.6.1 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::C_MINIMIZED = 1U << 3 [static, private]

Definition at line 72 of file Polyhedron.defs.hh.

Referenced by reset_c_minimized(), set_c_minimized(), and test_c_minimized().

11.207.6.2 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::C_UP_TO_DATE = 1U << 1 [static, private]

Definition at line 70 of file Polyhedron.defs.hh.

Referenced by reset_c_up_to_date(), set_c_up_to_date(), and test_c_up_to_date().

11.207.6.3 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::CS_PENDING = 1U << 7 [static, private]

Definition at line 76 of file Polyhedron.defs.hh.

Referenced by reset_c_pending(), set_c_pending(), and test_c_pending().

11.207.6.4 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::EMPTY = 1U << 0 [static, private]

Definition at line 69 of file Polyhedron.defs.hh.

Referenced by reset_empty(), reset_zero_dim_univ(), set_empty(), and test_empty().

11.207.6.5 flags_t Parma_Polyhedra_Library::Polyhedron::Status::flags [private]

This holds the current bitset.

Definition at line 81 of file Polyhedron.defs.hh.

Referenced by reset(), reset_zero_dim_univ(), set(), set_empty(), set_zero_dim_univ(), test_all(), test_any(), and test_zero_dim_univ().

11.207.6.6 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::G_MINIMIZED = 1U << 4 [static, private]

Definition at line 73 of file Polyhedron.defs.hh.

Referenced by reset_g_minimized(), set_g_minimized(), and test_g_minimized().

11.207.6.7 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::G_UP_TO_DATE = 1U << 2 [static, private]

Definition at line 71 of file Polyhedron.defs.hh.

Referenced by reset_g_up_to_date(), set_g_up_to_date(), and test_g_up_to_date().

11.207.6.8 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::GS_PENDING = 1U << 8 [static, private]

Definition at line 77 of file Polyhedron.defs.hh.

Referenced by reset_g_pending(), set_g_pending(), and test_g_pending().

11.207.6.9 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::SAT_C_UP_TO_DATE = 1U << 5 [static, private]

Definition at line 74 of file Polyhedron.defs.hh.

Referenced by reset_sat_c_up_to_date(), set_sat_c_up_to_date(), and test_sat_c_up_to_date().

11.207.6.10 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::SAT_G_UP_TO_DATE = 1U << 6 [static, private]

Definition at line 75 of file Polyhedron.defs.hh.

Referenced by reset_sat_g_up_to_date(), set_sat_g_up_to_date(), and test_sat_g_up_to_date().

11.207.6.11 const flags_t Parma_Polyhedra_Library::Polyhedron::Status::ZERO_DIM_UNIV = 0U [static, private]

Definition at line 68 of file Polyhedron.defs.hh.

Referenced by reset_zero_dim_univ(), set_zero_dim_univ(), and test_zero_dim_univ().

The documentation for this class was generated from the following files:

- Polyhedron.defs.hh
- Ph_Status.inlines.hh
- Ph_Status.cc

11.208 Parma_Polyhedra_Library::Octagonal_Shape< T >::Status Class Reference

A conjunctive assertion about a Octagonal_Shape<T> object.

Public Member Functions

- Status ()

  By default Status is the zero-dim universe assertion.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump (std::ostream &s) const

  Writes to `s` an ASCII representation of the internal representation of `*this`.

- bool ascii_load (std::istream &s)

  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Test, remove or add an individual assertion from the conjunction.

- bool test_zero_dim_univ () const
- void reset_zero_dim_univ ()
- void set_zero_dim_univ ()
- bool test_empty () const
- void reset_empty ()
- void set_empty ()
- bool test_strongly_closed () const
- void reset_strongly_closed ()
- void set_strongly_closed ()

Private Types

- typedef unsigned int flags_t

  Status is implemented by means of a finite bitset.

Private Member Functions

- Status (flags_t mask)

  Construct from a bitmask.

- bool test_all (flags_t mask) const

  Check whether all bits in `mask` *are set.*

- bool test_any (flags_t mask) const

  Check whether at least one bit in `mask` *is set.*

- void set (flags_t mask)

  Set the bits in `mask`*.*

- void reset (flags_t mask)

  Reset the bits in `mask`*.*

Private Attributes

- flags_t flags

  This holds the current bitset.

Static Private Attributes

Bitmasks for the individual assertions.

- static const flags_t ZERO_DIM_UNIV = 0U
- static const flags_t EMPTY = 1U << 0
- static const flags_t STRONGLY_CLOSED = 1U << 1

11.208.1 Detailed Description

template<typename T> class Parma_Polyhedra_Library::Octagonal_Shape< T >::Status

A conjunctive assertion about a Octagonal_Shape<T> object. The assertions supported are:

- *zero-dim universe*: the polyhedron is the zero-dimensional vector space $\mathbb{R}^0 = \{\cdot\}$;
- *empty*: the polyhedron is the empty set;
- *strongly closed*: the Octagonal_Shape object is strongly closed, so that all the constraints are as tight as possible.

Not all the conjunctions of these elementary assertions constitute a legal Status. In fact:

- *zero-dim universe* excludes any other assertion;
- *empty*: excludes any other assertion.

Definition at line 47 of file Octagonal_Shape.defs.hh.

11.208.2 Member Typedef Documentation

11.208.2.1 template<typename T> typedef unsigned int Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::flags_t [private]

Status is implemented by means of a finite bitset.

Definition at line 40 of file Octagonal_Shape.defs.hh.

11.208.3 Constructor & Destructor Documentation

11.208.3.1 template<typename T> Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::Status ()

By default Status is the *zero-dim universe* assertion.

11.208.3.2 template<typename T> Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::Status (flags_t *mask*) [private]

Construct from a bitmask.

11.208.4 Member Function Documentation

11.208.4.1 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of the internal representation of `*this`.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::ascii_dump().

11.208.4.2 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

11.208.4.3 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::OK () const

Checks if all the invariants are satisfied.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::OK().

11.208.4.4 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::reset (flags_t *mask*) [private]

Reset the bits in `mask`.

11.208.4.5 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::reset_empty ()

11.208.4.6 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::reset_strongly_closed ()

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::reset_strongly_closed().

11.208.4.7 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::reset_zero_dim_univ ()

11.208.4.8 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set (flags_t *mask*) [private]

Set the bits in `mask`.

11.208.4.9 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_empty ()

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::set_empty().

11.208.4.10 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_strongly_closed ()

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::set_strongly_closed().

11.208.4.11 template<typename T> void Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::set_zero_dim_univ ()

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::set_zero_dim_univ().

11.208.4.12 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_all (flags_t *mask*) const [private]

Check whether *all* bits in `mask` are set.

11.208.4.13 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_any (flags_t *mask*) const [private]

Check whether *at least one* bit in `mask` is set.

11.208.4.14 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_empty () const

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_empty().

11.208.4.15 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_strongly_closed () const

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_strongly_closed().

11.208.4.16 template<typename T> bool Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::test_zero_dim_univ () const

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::marked_zero_dim_univ().

11.208.5 Member Data Documentation

11.208.5.1 template<typename T> const flags_t Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::EMPTY = 1U << 0 [static, private]

Definition at line 45 of file Octagonal_Shape.defs.hh.

11.208.5.2 template<typename T> flags_t Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::flags [private]

This holds the current bitset.

Definition at line 50 of file Octagonal_Shape.defs.hh.

11.208.5.3 template<typename T> const flags_t Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::STRONGLY_CLOSED = 1U << 1 [static, private]

Definition at line 46 of file Octagonal_Shape.defs.hh.

11.208.5.4 template<typename T> const flags_t Parma_Polyhedra_Library::Octagonal_Shape< T >::Status::ZERO_DIM_UNIV = 0U [static, private]

Definition at line 44 of file Octagonal_Shape.defs.hh.

The documentation for this class was generated from the following file:

- Octagonal_Shape.defs.hh

11.209 Parma_Polyhedra_Library::stdiobuf Class Reference

`#include <stdiobuf.defs.hh>`

Public Member Functions

- stdiobuf (FILE *file)
  Constructor.

Protected Member Functions

- virtual int_type underflow ()
  Gets a character in case of underflow.
- virtual int_type uflow ()
  In case of underflow, gets a character and advances the next pointer.
- virtual std::streamsize xsgetn (char_type *s, std::streamsize n)
  Gets a sequence of characters.
- virtual int_type pbackfail (int_type c=traits_type::eof())
  Puts character back in case of backup underflow.
- virtual std::streamsize xsputn (const char_type *s, std::streamsize n)
  Writes a sequence of characters.
- virtual int_type overflow (int_type c)
  Writes a character in case of overflow.
- virtual int sync ()
  Synchronizes the stream buffer.

Private Types

- typedef char char_type

  Character type of the streambuf.

- typedef std::char_traits< char_type > traits_type

  Traits type of the streambuf.

- typedef traits_type::int_type int_type

  Integer type of the streambuf.

Private Attributes

- FILE * fp

  The encapsulated stdio file.

- int_type ungetc_buf

  Buffer for the last character read.

11.209.1 Detailed Description

Definition at line 31 of file stdiobuf.defs.hh.

11.209.2 Member Typedef Documentation

11.209.2.1 typedef char Parma_Polyhedra_Library::stdiobuf::char_type [private]

Character type of the streambuf.

Definition at line 90 of file stdiobuf.defs.hh.

11.209.2.2 typedef traits_type::int_type Parma_Polyhedra_Library::stdiobuf::int_type [private]

Integer type of the streambuf.

Definition at line 96 of file stdiobuf.defs.hh.

11.209.2.3 typedef std::char_traits<char_type> Parma_Polyhedra_Library::stdiobuf::traits_type [private]

Traits type of the streambuf.

Definition at line 93 of file stdiobuf.defs.hh.

11.209.3 Constructor & Destructor Documentation

11.209.3.1 Parma_Polyhedra_Library::stdiobuf::stdiobuf (FILE * *file*) [inline]

Constructor.

Definition at line 30 of file stdiobuf.inlines.hh.

11.209.4 Member Function Documentation

11.209.4.1 stdiobuf::int_type Parma_Polyhedra_Library::stdiobuf::overflow (int_type *c*) [protected, virtual]

Writes a character in case of overflow.

Specified by ISO/IEC 14882:1998: 27.5.2.4.5.

Definition at line 65 of file stdiobuf.cc.

References fp.

11.209.4.2 stdiobuf::int_type Parma_Polyhedra_Library::stdiobuf::pbackfail (int_type *c* = traits_type::eof()) [protected, virtual]

Puts character back in case of backup underflow.

Remarks

Specified by ISO/IEC 14882:1998: 27.5.2.4.4.

Definition at line 52 of file stdiobuf.cc.

References fp, and ungetc_buf.

11.209.4.3 int Parma_Polyhedra_Library::stdiobuf::sync () [protected, virtual]

Synchronizes the stream buffer.

Specified by ISO/IEC 14882:1998: 27.5.2.4.2.

Definition at line 74 of file stdiobuf.cc.

References fp.

11.209.4.4 stdiobuf::int_type Parma_Polyhedra_Library::stdiobuf::uflow () [protected, virtual]

In case of underflow, gets a character and advances the next pointer.

Remarks

Specified by ISO/IEC 14882:1998: 27.5.2.4.3.

Definition at line 30 of file stdiobuf.cc.

References fp, and ungetc_buf.

11.209.4.5 stdiobuf::int_type Parma_Polyhedra_Library::stdiobuf::underflow () [protected, virtual]

Gets a character in case of underflow.

Remarks

Specified by ISO/IEC 14882:1998: 27.5.2.4.3.

Definition at line 36 of file stdiobuf.cc.

References fp.

11.209.4.6 std::streamsize Parma_Polyhedra_Library::stdiobuf::xsgetn (char_type * *s*, std::streamsize *n*) [protected, virtual]

Gets a sequence of characters.

Remarks

Specified by ISO/IEC 14882:1998: 27.5.2.4.3.

Definition at line 42 of file stdiobuf.cc.

References fp, and ungetc_buf.

11.209.4.7 std::streamsize Parma_Polyhedra_Library::stdiobuf::xsputn (const char_type * *s*, std::streamsize *n*) [protected, virtual]

Writes a sequence of characters.

Remarks

Specified by ISO/IEC 14882:1998: 27.5.2.4.5.

Definition at line 60 of file stdiobuf.cc.

References fp.

11.209.5 Member Data Documentation

11.209.5.1 FILE* Parma_Polyhedra_Library::stdiobuf::fp [private]

The encapsulated stdio file.

Definition at line 99 of file stdiobuf.defs.hh.

Referenced by overflow(), pbackfail(), sync(), uflow(), underflow(), xsgetn(), and xsputn().

11.209.5.2 int_type Parma_Polyhedra_Library::stdiobuf::ungetc_buf [private]

Buffer for the last character read.

Definition at line 102 of file stdiobuf.defs.hh.

Referenced by pbackfail(), uflow(), and xsgetn().

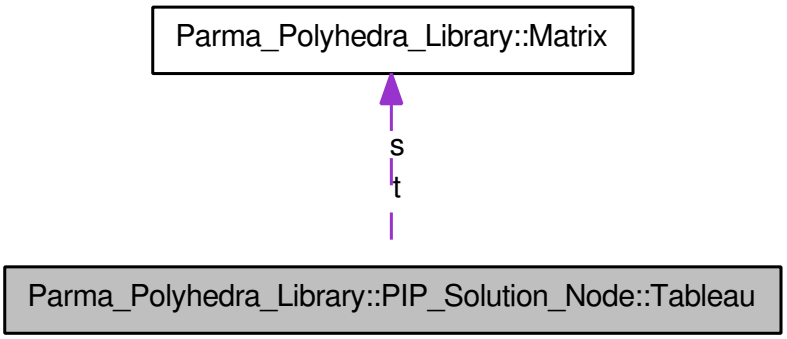
The documentation for this class was generated from the following files:

- stdiobuf.defs.hh
- stdiobuf.inlines.hh
- stdiobuf.cc

11.210 Parma_Polyhedra_Library::PIP_Solution_Node::Tableau Struct Reference

The type for parametric simplex tableau.

Collaboration diagram for Parma_Polyhedra_Library::PIP_Solution_Node::Tableau:



Public Member Functions

- Tableau ()

  Default constructor.

- Tableau (const Tableau &y)

  Copy constructor.

- ~Tableau ()
  Destructor.
- bool is_integer () const
  Tests whether the matrix is integer, i.e., the denominator is 1.
- void scale (Coefficient_traits::const_reference ratio)
  Multiplies all coefficients and denominator with ratio.
- void normalize ()
  Normalizes the modulo of coefficients so that they are mutually prime.
- bool is_better_pivot (const std::vector< dimension_type > &mapping, const std::vector< bool > &basis, const dimension_type i, const dimension_type j, const dimension_type i_, const dimension_type j_) const
  Compares two pivot row and column pairs before pivoting.
- Coefficient_traits::const_reference denominator () const
  Returns the value of the denominator.
- void ascii_dump (std::ostream &s) const
  Dumps to `s` an ASCII representation of `*this`.
- bool ascii_load (std::istream &s)
  Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.
- memory_size_type external_memory_in_bytes () const
  Returns the size in bytes of the memory managed by `*this`.
- bool OK () const
  Returns `true` if and only if `*this` is well formed.

Public Attributes

- Matrix s
  The matrix of simplex coefficients.
- Matrix t
  The matrix of parameter coefficients.
- Coefficient denom
  A common denominator for all matrix elements.

11.210.1 Detailed Description

The type for parametric simplex tableau.

Definition at line 393 of file PIP_Tree.defs.hh.

11.210.2 Constructor & Destructor Documentation

11.210.2.1 Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::Tableau () [inline]

Default constructor.

Definition at line 30 of file PIP_Tree.inlines.hh.

References OK().

11.210.2.2 Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::Tableau (const Tableau & *y*) [inline]

Copy constructor.

Definition at line 36 of file PIP_Tree.inlines.hh.

References OK().

11.210.2.3 Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::~Tableau () [inline]

Destructor.

Definition at line 42 of file PIP_Tree.inlines.hh.

11.210.3 Member Function Documentation

11.210.3.1 void Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::ascii_dump (std::ostream & *s*) const

Dumps to `s` an ASCII representation of `*this`.

Definition at line 1232 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Matrix::ascii_dump(), denom, s, and t.

11.210.3.2 bool Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 1241 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Matrix::ascii_load(), denom, OK(), s, and t.

11.210.3.3 Coefficient_traits::const_reference Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::denominator () const [inline]

Returns the value of the denominator.

Definition at line 51 of file PIP_Tree.inlines.hh.

References denom.

11.210.3.4 memory_size_type Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::external_memory_in_bytes () const

Returns the size in bytes of the memory managed by `*this`.

Note

No need for a `total_memory_in_bytes()` method, since class Tableau is a private inner class of PIP_Solution_Node.

Definition at line 2878 of file PIP_Tree.cc.

References denom, Parma_Polyhedra_Library::Matrix::external_memory_in_bytes(), s, and t.

11.210.3.5 bool Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot (const std::vector< dimension_type > & *mapping*, const std::vector< bool > & *basis*, const dimension_type *i*, const dimension_type *j*, const dimension_type *i_*, const dimension_type *j_*) const

Compares two pivot row and column pairs before pivoting.

The algorithm searches the first (ie, leftmost) column k in parameter matrix for which the $c = s_{*j}\frac{t_{ik}}{s_{ij}}$ and $c' = s_{*j'}\frac{t_{i'k}}{s_{i'j'}}$ columns are different, where s_{*j} denotes the j^{th} column from the s matrix and $s_{*j'}$ is the j'^{th} column of s.

c is the computed column that would be subtracted to column k in parameter matrix if pivoting is done using the (i, j) row and column pair. c' is the computed column that would be subtracted to column k in parameter matrix if pivoting is done using the (i', j') row and column pair.

The test is true if the computed $-c$ column is lexicographically bigger than the $-c'$ column. Due to the column ordering in the parameter matrix of the tableau, leftmost search will enforce solution increase with respect to the following priority order:

- the constant term
- the coefficients for the original parameters
- the coefficients for the oldest artificial parameters.

Returns

`true` if pivot row and column pair (i, j) is more suitable for pivoting than the (i', j') pair

Parameters

mapping the PIP_Solution_Node::mapping vector for the tableau

basis the PIP_Solution_Node::basis vector for the tableau

i the row number for the first pivot row and column pair to be compared

j the column number for the first pivot row and column pair to be compared

i_ the row number for the second pivot row and column pair to be compared

j_ the column number for the second pivot row and column pair to be compared

Definition at line 1140 of file PIP_Tree.cc.

References Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, s, t, WEIGHT_ADD, and WEIGHT_BEGIN.

11.210.3.6 bool Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_integer () const [inline]

Tests whether the matrix is integer, i.e., the denominator is 1.

Definition at line 46 of file PIP_Tree.inlines.hh.

References denom.

11.210.3.7 void Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize ()

Normalizes the modulo of coefficients so that they are mutually prime.

Computes the Greatest Common Divisor (GCD) among the elements of the matrices and normalizes them and the denominator by the GCD itself.

Definition at line 1064 of file PIP_Tree.cc.

References denom, Parma_Polyhedra_Library::exact_div_assign(), Parma_Polyhedra_Library::gcd_assign(), Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), PPL_DIRTY_TEMP_COEFFICIENT, s, t, WEIGHT_ADD, WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

11.210.3.8 bool Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::OK () const

Returns `true` if and only if `*this` is well formed.

Definition at line 609 of file PIP_Tree.cc.

References denom, Parma_Polyhedra_Library::Matrix::num_rows(), Parma_Polyhedra_Library::Matrix::OK(), s, and t.

Referenced by ascii_load(), and Tableau().

11.210.3.9 void Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::scale (Coefficient_traits::const_reference *ratio*)

Multiplies all coefficients and denominator with ratio.

Definition at line 1121 of file PIP_Tree.cc.

References denom, Parma_Polyhedra_Library::Matrix::num_columns(), Parma_Polyhedra_Library::Matrix::num_rows(), s, t, WEIGHT_ADD_MUL, and WEIGHT_BEGIN.

11.210.4 Member Data Documentation

11.210.4.1 Coefficient Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::denom

A common denominator for all matrix elements.

Definition at line 399 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), denominator(), external_memory_in_bytes(), is_integer(), normalize(), OK(), and scale().

11.210.4.2 Matrix Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::s

The matrix of simplex coefficients.

Definition at line 395 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), is_better_pivot(), normalize(), OK(), and scale().

11.210.4.3 Matrix Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::t

The matrix of parameter coefficients.

Definition at line 397 of file PIP_Tree.defs.hh.

Referenced by ascii_dump(), ascii_load(), external_memory_in_bytes(), is_better_pivot(), normalize(), OK(), and scale().

The documentation for this struct was generated from the following files:

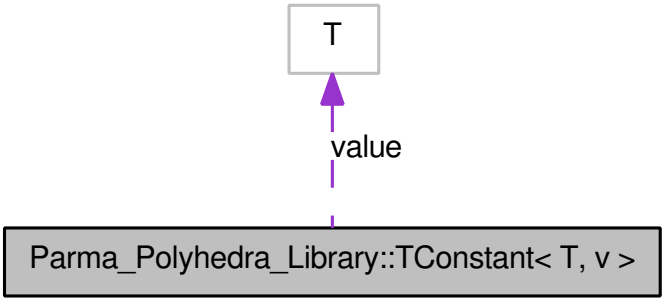
- PIP_Tree.defs.hh
- PIP_Tree.inlines.hh
- PIP_Tree.cc

11.211 Parma_Polyhedra_Library::TConstant< T, v > Struct Template Reference

`#include <globals.defs.hh>`

Inheritance diagram for Parma_Polyhedra_Library::TConstant< T, v >:

Collaboration diagram for Parma_Polyhedra_Library::TConstant< T, v >:



Static Public Attributes

- static const T value = v

11.211.1 Detailed Description

template<typename T, long long v> struct Parma_Polyhedra_Library::TConstant< T, v >

Definition at line 351 of file globals.defs.hh.

11.211.2 Member Data Documentation

11.211.2.1 template<typename T, long long v> const T Parma_Polyhedra_Library::TConstant< T, v >::value = v [inline, static]

Definition at line 352 of file globals.defs.hh.

The documentation for this struct was generated from the following file:

- globals.defs.hh

11.212 Parma_Polyhedra_Library::Throwable Class Reference

User objects the PPL can throw.

`#include <globals.defs.hh>`

Public Member Functions

- virtual void throw_me () const =0
  Throws the user defined exception object.
- virtual ~Throwable ()
  Virtual destructor.

Related Functions

(Note that these are not member functions.)

- void maybe_abandon ()
  If the pointer abandon_expensive_computations is found to be nonzero, the exception it points to is thrown.

11.212.1 Detailed Description

User objects the PPL can throw. This abstract base class should be instantiated by those users willing to provide a polynomial upper bound to the time spent by any invocation of a library operator.

Definition at line 183 of file globals.defs.hh.

11.212.2 Constructor & Destructor Documentation

11.212.2.1 Parma_Polyhedra_Library::Throwable::~Throwable () [inline, virtual]

Virtual destructor.

Definition at line 53 of file globals.inlines.hh.

11.212.3 Member Function Documentation

11.212.3.1 virtual void Parma_Polyhedra_Library::Throwable::throw_me () const [pure virtual]

Throws the user defined exception object.

11.212.4 Friends And Related Function Documentation

11.212.4.1 void maybe_abandon () [related]

If the pointer abandon_expensive_computations is found to be nonzero, the exception it points to is thrown.

Definition at line 57 of file globals.inlines.hh.

The documentation for this class was generated from the following files:

- globals.defs.hh
- globals.inlines.hh

11.213 Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign Class Reference

Scalar product sign function object depending on topology.

`#include <Scalar_Products.defs.hh>`

Public Member Functions

- Topology_Adjusted_Scalar_Product_Sign (const Constraint &c)
  Constructs the function object according to the topology of c.
- Topology_Adjusted_Scalar_Product_Sign (const Generator &g)
  Constructs the function object according to the topology of g.
- int operator() (const Constraint &, const Generator &) const
  Computes the (topology adjusted) scalar product sign of c and g.
- int operator() (const Generator &, const Constraint &) const
  Computes the (topology adjusted) scalar product sign of g and c.

Private Types

- typedef int(* SPS_type)(const Linear_Row &, const Linear_Row &)
  The type of the scalar product sign function pointer.

Private Attributes

- SPS_type sps_fp
  The scalar product sign function pointer.

11.213.1 Detailed Description

Scalar product sign function object depending on topology.

Definition at line 176 of file Scalar_Products.defs.hh.

11.213.2 Member Typedef Documentation

11.213.2.1 typedef int(* Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::SPS_type)(const Linear_Row &, const Linear_Row &) [private]

The type of the scalar product sign function pointer.

Definition at line 190 of file Scalar_Products.defs.hh.

11.213.3 Constructor & Destructor Documentation

11.213.3.1 Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::Topology_Adjusted_Scalar_Product_Sign (const Constraint & *c*) [inline]

Constructs the function object according to the topology of `c`.

Definition at line 130 of file Scalar_Products.inlines.hh.

11.213.3.2 Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::Topology_Adjusted_Scalar_Product_Sign (const Generator & *g*) [inline]

Constructs the function object according to the topology of `g`.

Definition at line 138 of file Scalar_Products.inlines.hh.

11.213.4 Member Function Documentation

11.213.4.1 int Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator() (const Generator & *g*, const Constraint & *c*) const [inline]

Computes the (topology adjusted) scalar product sign of `g` and `c`.

Definition at line 156 of file Scalar_Products.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Constraint::space_dimension(), Parma_Polyhedra_Library::Generator::space_dimension(), and sps_fp.

11.213.4.2 int Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign::operator() (const Constraint & *c*, const Generator & *g*) const [inline]

Computes the (topology adjusted) scalar product sign of `c` and `g`.

Definition at line 145 of file Scalar_Products.inlines.hh.

References Parma_Polyhedra_Library::Linear_Row::is_necessarily_closed(), Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), Parma_Polyhedra_Library::Scalar_Products::sign(), Parma_Polyhedra_Library::Generator::space_dimension(), Parma_Polyhedra_Library::Constraint::space_dimension(), and sps_fp.

11.213.5 Member Data Documentation

11.213.5.1 SPS_type Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_-Sign::sps_fp [private]

The scalar product sign function pointer.

Definition at line 193 of file Scalar_Products.defs.hh.

Referenced by operator()().

The documentation for this class was generated from the following files:

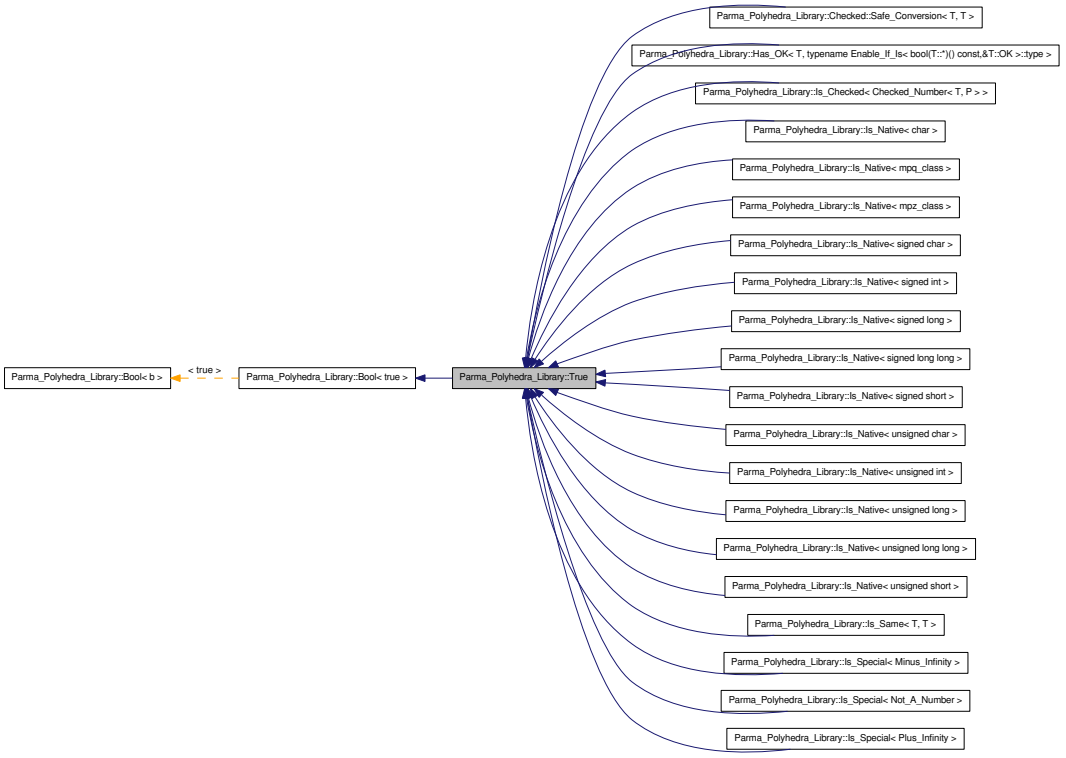
- Scalar_Products.defs.hh
- Scalar_Products.inlines.hh

11.214 Parma_Polyhedra_Library::True Struct Reference

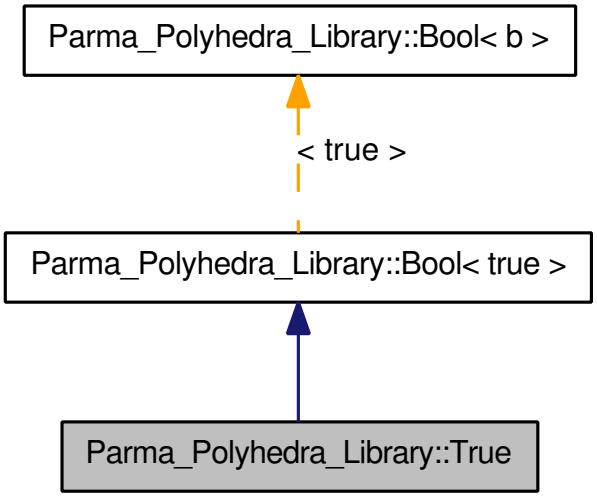
A class holding a constant called `value` that evaluates to `true`.

`#include <meta_programming.hh>`

Inheritance diagram for Parma_Polyhedra_Library::True:



Collaboration diagram for Parma_Polyhedra_Library::True:



11.214.1 Detailed Description

A class holding a constant called `value` that evaluates to `true`.

Definition at line 157 of file meta_programming.hh.

The documentation for this struct was generated from the following file:

- meta_programming.hh

11.215 Parma_Polyhedra_Library::Variable Class Reference

A dimension of the vector space.

`#include <Variable.defs.hh>`

Classes

- struct Compare

  Binary predicate defining the total ordering on variables.

Public Types

- typedef void output_function_type (std::ostream &s, const Variable &v)

  Type of output functions.

Public Member Functions

- Variable (dimension_type i)

Builds the variable corresponding to the Cartesian axis of index `i`.

- dimension_type id () const

  Returns the index of the Cartesian axis associated to the variable.

- dimension_type space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Variable can handle.

- static void set_output_function (output_function_type *p)

  Sets the output function to be used for printing Variable objects.

- static output_function_type * get_output_function ()

  Returns the pointer to the current output function.

Static Private Member Functions

- static void default_output_function (std::ostream &s, const Variable &v)

  The default output function.

Private Attributes

- dimension_type varid

  The index of the Cartesian axis.

Static Private Attributes

- static output_function_type * current_output_function = 0

  Pointer to the current output function.

Friends

- class Init
- std::ostream & Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream &s, const Variable &v)

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Variable &v)

  Output operator.

- bool less (Variable v, Variable w)

  Defines a total ordering on variables.

11.215.1 Detailed Description

A dimension of the vector space. An object of the class Variable represents a dimension of the space, that is one of the Cartesian axes. Variables are used as basic blocks in order to build more complex linear expressions. Each variable is identified by a non-negative integer, representing the index of the corresponding Cartesian axis (the first axis has index 0). The space dimension of a variable is the dimension of the vector space made by all the Cartesian axes having an index less than or equal to that of the considered variable; thus, if a variable has index i, its space dimension is $i + 1$.

Note that the "meaning" of an object of the class Variable is completely specified by the integer index provided to its constructor: be careful not to be mislead by C++ language variable names. For instance, in the following example the linear expressions `e1` and `e2` are equivalent, since the two variables `x` and `z` denote the same Cartesian axis.

```
Variable x(0);
Variable y(1);
Variable z(0);
Linear_Expression e1 = x + y;
Linear_Expression e2 = y + z;
```

Definition at line 81 of file Variable.defs.hh.

11.215.2 Member Typedef Documentation

11.215.2.1 typedef void Parma_Polyhedra_Library::Variable::output_function_type(std::ostream &s, const Variable &v)

Type of output functions.

Definition at line 114 of file Variable.defs.hh.

11.215.3 Constructor & Destructor Documentation

11.215.3.1 Parma_Polyhedra_Library::Variable::Variable (dimension_type *i*) [inline, explicit]

Builds the variable corresponding to the Cartesian axis of index `i`.

Exceptions

std::length_error Thrown if `i+1` exceeds `Variable::max_space_dimension()`.

Definition at line 38 of file Variable.inlines.hh.

11.215.4 Member Function Documentation

11.215.4.1 void Parma_Polyhedra_Library::Variable::default_output_function (std::ostream & *s*, const Variable & *v*) [static, private]

The default output function.

Definition at line 39 of file Variable.cc.

References id(), and varid.

Referenced by Parma_Polyhedra_Library::Init::Init().

11.215.4.2 memory_size_type Parma_Polyhedra_Library::Variable::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 57 of file Variable.inlines.hh.

Referenced by total_memory_in_bytes().

11.215.4.3 Variable::output_function_type * Parma_Polyhedra_Library::Variable::get_output_function () [inline, static]

Returns the pointer to the current output function.

Definition at line 72 of file Variable.inlines.hh.

References current_output_function.

11.215.4.4 dimension_type Parma_Polyhedra_Library::Variable::id () const [inline]

Returns the index of the Cartesian axis associated to the variable.

Definition at line 47 of file Variable.inlines.hh.

References varid.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Congruence::coefficient(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_Library::Grid::constrains(), default_output_function(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Octagonal_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Polyhedron::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Grid::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_Polyhedra_Library::BD_Shape< T >::incremental_shortest_path_closure_assign(), Parma_Polyhedra_Library::Octagonal_Shape< T >::incremental_strong_closure_assign(), Parma_Polyhedra_Library::Variables_Set::insert(), less(), OK(), Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), and Parma_Polyhedra_Library::Variables_Set::Variables_Set().

11.215.4.5 dimension_type Parma_Polyhedra_Library::Variable::max_space_dimension () [inline, static]

Returns the maximum space dimension a Variable can handle.

Definition at line 33 of file Variable.inlines.hh.

References Parma_Polyhedra_Library::not_a_dimension().

Referenced by OK().

11.215.4.6 bool Parma_Polyhedra_Library::Variable::OK () const

Checks if all the invariants are satisfied.

Definition at line 34 of file Variable.cc.

References id(), and max_space_dimension().

11.215.4.7 void Parma_Polyhedra_Library::Variable::set_output_function (output_function_type *p) [inline, static]

Sets the output function to be used for printing Variable objects.

Definition at line 67 of file Variable.inlines.hh.

References current_output_function.

Referenced by Parma_Polyhedra_Library::Init::Init().

11.215.4.8 dimension_type Parma_Polyhedra_Library::Variable::space_dimension () const [inline]

Returns the dimension of the vector space enclosing `*this`.

The returned value is `id()+1`.

Definition at line 52 of file Variable.inlines.hh.

References varid.

Referenced by Parma_Polyhedra_Library::Polyhedron::affine_image(), Parma_Polyhedra_Library::Grid::affine_image(), Parma_Polyhedra_Library::Box< ITV >::affine_image(), Parma_Polyhedra_Library::Polyhedron::affine_preimage(), Parma_Polyhedra_Library::Grid::affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::affine_preimage(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_image(), Parma_Polyhedra_Library::Grid::bounded_affine_image(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_image(), Parma_Polyhedra_Library::Polyhedron::bounded_affine_preimage(), Parma_Polyhedra_Library::Grid::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::Linear_Expression::coefficient(), Parma_Polyhedra_Library::Grid_Generator::coefficient(), Parma_Polyhedra_Library::Generator::coefficient(), Parma_Polyhedra_Library::Constraint::coefficient(), Parma_Polyhedra_Library::Congruence::coefficient(), Parma_Polyhedra_Library::Polyhedron::constrains(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::constrains(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constrains(), Parma_Polyhedra_Library::Grid::constrains(), Parma_Polyhedra_Library::Box< ITV >::constrains(), Parma_Polyhedra_Library::BD_Shape< T >::constrains(), Parma_Polyhedra_Library::Polyhedron::expand_space_dimension(), Parma_Polyhedra_Library::Grid::expand_space_dimension(), Parma_Polyhedra_Library::Box< ITV >::expand_space_dimension(), Parma_Polyhedra_Library::BD_Shape< T >::expand_space_dimension(), Parma_Polyhedra_Library::Polyhedron::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Grid::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::get_interval(), Parma_Polyhedra_Library::Linear_Expression::Linear_Expression(), Parma_Polyhedra_Library::PIP_Solution_Node::parametric_values(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_Library::Box< ITV >::set_interval(), Parma_Polyhedra_Library::Polyhedron::throw_-

dimension_incompatible(), Parma_Polyhedra_Library::Grid::throw_dimension_incompatible(), Parma_Polyhedra_Library::Generator::throw_dimension_incompatible(), Parma_Polyhedra_Library::Constraint::throw_dimension_incompatible(), Parma_Polyhedra_Library::Congruence::throw_dimension_incompatible(), Parma_Polyhedra_Library::Polyhedron::unconstrain(), Parma_Polyhedra_Library::Grid::unconstrain(), and Parma_Polyhedra_Library::BD_Shape< T >::unconstrain().

11.215.4.9 memory_size_type Parma_Polyhedra_Library::Variable::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 62 of file Variable.inlines.hh.

References external_memory_in_bytes().

11.215.5 Friends And Related Function Documentation

11.215.5.1 friend class Init [friend]

Definition at line 134 of file Variable.defs.hh.

11.215.5.2 bool less (Variable *v*, Variable *w*) [related]

Defines a total ordering on variables.

Definition at line 78 of file Variable.inlines.hh.

References id().

Referenced by Parma_Polyhedra_Library::Variable::Compare::operator()().

11.215.5.3 std::ostream & operator<< (std::ostream & *s*, const Variable & *v*) [related]

Output operator.

Definition at line 48 of file Variable.cc.

11.215.5.4 std::ostream& Parma_Polyhedra_Library::IO_Operators::operator<< (std::ostream & *s*, const Variable & *v*) [friend]

11.215.6 Member Data Documentation

11.215.6.1 PPL::Variable::output_function_type * Parma_Polyhedra_Library::Variable::current_output_function = 0 [static, private]

Pointer to the current output function.

Definition at line 141 of file Variable.defs.hh.

Referenced by get_output_function(), and set_output_function().

11.215.6.2 dimension_type Parma_Polyhedra_Library::Variable::varid [private]

The index of the Cartesian axis.

Definition at line 131 of file Variable.defs.hh.

Referenced by default_output_function(), id(), and space_dimension().

The documentation for this class was generated from the following files:

- Variable.defs.hh
- Variable.inlines.hh
- Variable.cc

11.216 Parma_Polyhedra_Library::Variables_Set Class Reference

An std::set of variables' indexes.

`#include <Variables_Set.defs.hh>`

Public Member Functions

- Variables_Set ()

  Builds the empty set of variable indexes.

- Variables_Set (const Variable &v)

  Builds the singleton set of indexes containing `v.id()`;.

- Variables_Set (const Variable &v, const Variable &w)

  Builds the set of variables's indexes in the range from `v.id()` to `w.id()`.

- dimension_type space_dimension () const

  Returns the dimension of the smallest vector space enclosing all the variables whose indexes are in the set.

- void insert (Variable v)

  Inserts the index of variable `v` into the set.

- bool ascii_load (std::istream &s)

Loads from `s` *an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets* `*this` *accordingly. Returns* `true` *if successful,* `false` *otherwise.*

- memory_size_type total_memory_in_bytes () const

  Returns the total size in bytes of the memory occupied by `*this`.

- memory_size_type external_memory_in_bytes () const

  Returns the size in bytes of the memory managed by `*this`.

- bool OK () const

  Checks if all the invariants are satisfied.

- void ascii_dump () const

  Writes to `std::cerr` *an ASCII representation of* `*this`.

- void ascii_dump (std::ostream &s) const

  Writes to `s` *an ASCII representation of* `*this`.

- void print () const

  Prints `*this` *to* `std::cerr` *using* `operator<<`.

Static Public Member Functions

- static dimension_type max_space_dimension ()

  Returns the maximum space dimension a Variables_Set can handle.

Private Types

- typedef std::set< dimension_type > Base

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Variables_Set &v)

  Output operator.

11.216.1 Detailed Description

An std::set of variables' indexes.

Definition at line 47 of file Variables_Set.defs.hh.

11.216.2 Member Typedef Documentation

11.216.2.1 typedef std::set<dimension_type> Parma_Polyhedra_Library::Variables_Set::Base [private]

Definition at line 50 of file Variables_Set.defs.hh.

11.216.3 Constructor & Destructor Documentation

11.216.3.1 Parma_Polyhedra_Library::Variables_Set::Variables_Set () [inline]

Builds the empty set of variable indexes.

Definition at line 33 of file Variables_Set.inlines.hh.

11.216.3.2 Parma_Polyhedra_Library::Variables_Set::Variables_Set (const Variable & *v*) [inline, explicit]

Builds the singleton set of indexes containing `v.id()`;.

Definition at line 43 of file Variables_Set.inlines.hh.

References insert().

11.216.3.3 Parma_Polyhedra_Library::Variables_Set::Variables_Set (const Variable & *v*, const Variable & *w*)

Builds the set of variables's indexes in the range from `v.id()` to `w.id()`.

If `v.id() <= w.id()`, this constructor builds the set of variables' indexes `v.id()`, `v.id()+1`, ..., `w.id()`. The empty set is built otherwise.

Definition at line 30 of file Variables_Set.cc.

References Parma_Polyhedra_Library::Variable::id(), and insert().

11.216.4 Member Function Documentation

11.216.4.1 void Parma_Polyhedra_Library::Variables_Set::ascii_dump (std::ostream & *s*) const

Writes to `s` an ASCII representation of `*this`.

Definition at line 59 of file Variables_Set.cc.

11.216.4.2 void Parma_Polyhedra_Library::Variables_Set::ascii_dump () const

Writes to `std::cerr` an ASCII representation of `*this`.

Referenced by Parma_Polyhedra_Library::PIP_Problem::ascii_dump(), and Parma_Polyhedra_Library::MIP_Problem::ascii_dump().

11.216.4.3 bool Parma_Polyhedra_Library::Variables_Set::ascii_load (std::istream & *s*)

Loads from `s` an ASCII representation (as produced by ascii_dump(std::ostream&) const) and sets `*this` accordingly. Returns `true` if successful, `false` otherwise.

Definition at line 70 of file Variables_Set.cc.

11.216.4.4 memory_size_type Parma_Polyhedra_Library::Variables_Set::external_memory_in_bytes () const [inline]

Returns the size in bytes of the memory managed by `*this`.

Definition at line 60 of file Variables_Set.inlines.hh.

Referenced by total_memory_in_bytes().

11.216.4.5 void Parma_Polyhedra_Library::Variables_Set::insert (Variable *v*) [inline]

Inserts the index of variable `v` into the set.

Definition at line 38 of file Variables_Set.inlines.hh.

References Parma_Polyhedra_Library::Variable::id().

Referenced by Parma_Polyhedra_Library::PIP_Problem::add_space_dimensions_and_embed(), Parma_Polyhedra_Library::MIP_Problem::add_to_integer_space_dimensions(), Parma_Polyhedra_Library::PIP_Problem::add_to_parameter_space_dimensions(), Parma_Polyhedra_Library::MIP_Problem::choose_branching_variable(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), and Variables_Set().

11.216.4.6 dimension_type Parma_Polyhedra_Library::Variables_Set::max_space_dimension () [inline, static]

Returns the maximum space dimension a Variables_Set can handle.

Definition at line 49 of file Variables_Set.inlines.hh.

11.216.4.7 bool Parma_Polyhedra_Library::Variables_Set::OK () const

Checks if all the invariants are satisfied.

Definition at line 37 of file Variables_Set.cc.

Referenced by Parma_Polyhedra_Library::PIP_Problem::OK().

11.216.4.8 void Parma_Polyhedra_Library::Variables_Set::print () const

Prints `*this` to `std::cerr` using `operator<<`.

11.216.4.9 dimension_type Parma_Polyhedra_Library::Variables_Set::space_dimension () const [inline]

Returns the dimension of the smallest vector space enclosing all the variables whose indexes are in the set.

Definition at line 54 of file Variables_Set.inlines.hh.

Referenced by Parma_Polyhedra_Library::MIP_Problem::add_to_integer_space_dimensions(), Parma_Polyhedra_Library::PIP_Problem::add_to_parameter_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Grid::drop_some_non_integer_points(), Parma_Polyhedra_Library::Box< ITV >::drop_some_non_integer_points(), Parma_Polyhedra_Library::BD_Shape< T >::drop_some_non_integer_points(), Parma_Polyhedra_Library::Polyhedron::fold_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::Grid::fold_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::fold_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::fold_space_dimensions(), Parma_Polyhedra_Library::MIP_Problem::MIP_Problem(), Parma_Polyhedra_Library::PIP_Problem::PIP_Problem(), Parma_Polyhedra_Library::Polyhedron::remove_space_dimensions(), Parma_Polyhedra_Library::Octagonal_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Grid_Generator_System::remove_space_dimensions(), Parma_Polyhedra_Library::Grid::remove_space_dimensions(), Parma_Polyhedra_Library::Box< ITV >::remove_space_dimensions(), Parma_Polyhedra_Library::BD_Shape< T >::remove_space_dimensions(), Parma_Polyhedra_Library::Polyhedron::unconstrain(), Parma_Polyhedra_Library::Octagonal_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Grid::unconstrain(), Parma_Polyhedra_Library::Box< ITV >::unconstrain(), Parma_Polyhedra_Library::BD_Shape< T >::unconstrain(), Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

11.216.4.10 memory_size_type Parma_Polyhedra_Library::Variables_Set::total_memory_in_bytes () const [inline]

Returns the total size in bytes of the memory occupied by `*this`.

Definition at line 69 of file Variables_Set.inlines.hh.

References external_memory_in_bytes().

11.216.5 Friends And Related Function Documentation

11.216.5.1 std::ostream & operator<< (std::ostream & *s*, const Variables_Set & *vs*) [related]

Output operator.

Definition at line 46 of file Variables_Set.cc.

The documentation for this class was generated from the following files:

- Variables_Set.defs.hh
- Variables_Set.inlines.hh
- Variables_Set.cc

11.217 Parma_Polyhedra_Library::Weightwatch_Traits Struct Reference

Traits class for the deterministic timeout mechanism.

`#include <globals.defs.hh>`

Public Types

- typedef unsigned long long Threshold

  The type used to specify thresholds for computational weight.

- typedef unsigned long long Delta

  The type used to specify increments of computational weight.

Static Public Member Functions

- static const Threshold & get ()

  Returns the current computational weight.

- static bool less_than (const Threshold &a, const Threshold &b)

  Compares the two weights `a` and `b`.

- static void from_delta (Threshold &threshold, const Delta &delta)

  Sets `threshold` to be `delta` units bigger than the current weigth.

Static Public Attributes

- static Threshold weight = 0

  The current computational weight.

- static void(* check_function)(void)

  A pointer to the function that has to be called when checking the reaching of thresholds.

11.217.1 Detailed Description

Traits class for the deterministic timeout mechanism. This abstract base class should be instantiated by those users willing to provide a polynomial upper bound to the time spent by any invocation of a library operator.

Definition at line 97 of file globals.defs.hh.

11.217.2 Member Typedef Documentation

11.217.2.1 typedef unsigned long long Parma_Polyhedra_Library::Weightwatch_Traits::Delta

The type used to specify increments of computational weight.

Definition at line 102 of file globals.defs.hh.

11.217.2.2 typedef unsigned long long Parma_Polyhedra_Library::Weightwatch_-Traits::Threshold

The type used to specify thresholds for computational weight.

Definition at line 99 of file globals.defs.hh.

11.217.3 Member Function Documentation

11.217.3.1 void Parma_Polyhedra_Library::Weightwatch_Traits::from_delta (Threshold & *threshold*, const Delta & *delta*) [inline, static]

Sets `threshold` to be `delta` units bigger than the current weigth.

Definition at line 48 of file globals.inlines.hh.

References weight.

11.217.3.2 const Weightwatch_Traits::Threshold & Parma_Polyhedra_Library::Weightwatch_-Traits::get () [inline, static]

Returns the current computational weight.

Definition at line 38 of file globals.inlines.hh.

References weight.

11.217.3.3 bool Parma_Polyhedra_Library::Weightwatch_Traits::less_than (const Threshold & *a*, const Threshold & *b*) [inline, static]

Compares the two weights `a` and `b`.

Definition at line 43 of file globals.inlines.hh.

11.217.4 Member Data Documentation

11.217.4.1 void(∗ Parma_Polyhedra_Library::Weightwatch_Traits::check_function)(void)=0 [static]

A pointer to the function that has to be called when checking the reaching of thresholds.

The pointer can be null if no thresholds are set.

Definition at line 122 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::maybe_abandon().

11.217.4.2 Weightwatch_Traits::Threshold Parma_Polyhedra_Library::Weightwatch_Traits::weight = 0 [static]

The current computational weight.

Definition at line 114 of file globals.defs.hh.

Referenced by from_delta(), and get().

The documentation for this struct was generated from the following files:

- globals.defs.hh
- globals.inlines.hh
- globals.cc

11.218 Parma_Polyhedra_Library::Widening_Function< PSET > Class Template Reference

Wraps a widening method into a function object.

`#include <Widening_Function.defs.hh>`

Public Types

- typedef void(PSET::∗ Widening_Method)(const PSET &, unsigned ∗)

  The (parametric) type of a widening method.

Public Member Functions

- Widening_Function (Widening_Method wm)

  Explicit unary constructor.

- void operator() (PSET &x, const PSET &y, unsigned ∗tp=0) const

Function-application operator.

Private Attributes

- Widening_Method w_method

  The widening method.

11.218.1 Detailed Description

template<typename PSET> class Parma_Polyhedra_Library::Widening_Function< PSET >

Wraps a widening method into a function object.

Definition at line 35 of file Widening_Function.defs.hh.

11.218.2 Member Typedef Documentation

11.218.2.1 template<typename PSET> typedef void(PSET::* Parma_Polyhedra_Library::Widening_Function< PSET >::Widening_Method)(const PSET &, unsigned *)

The (parametric) type of a widening method.

Definition at line 31 of file Widening_Function.defs.hh.

11.218.3 Constructor & Destructor Documentation

11.218.3.1 template<typename PSET > Parma_Polyhedra_Library::Widening_Function< PSET >::Widening_Function (Widening_Method *wm*) [inline, explicit]

Explicit unary constructor.

Definition at line 32 of file Widening_Function.inlines.hh.

11.218.4 Member Function Documentation

11.218.4.1 template<typename PSET > void Parma_Polyhedra_Library::Widening_Function< PSET >::operator() (PSET & *x*, const PSET & *y*, unsigned * *tp* = 0) const [inline]

Function-application operator.

Computes `(x.*wm)(y, tp)`, where `wm` is the widening method stored at construction time.

Definition at line 39 of file Widening_Function.inlines.hh.

References Parma_Polyhedra_Library::Widening_Function< PSET >::w_method.

11.218.5 Member Data Documentation

11.218.5.1 template<typename PSET> Widening_Method Parma_Polyhedra_Library::Widening_Function< PSET >::w_method [private]

The widening method.

Definition at line 46 of file Widening_Function.defs.hh.

Referenced by Parma_Polyhedra_Library::Widening_Function< PSET >::operator()().

The documentation for this class was generated from the following files:

- Widening_Function.defs.hh
- Widening_Function.inlines.hh

11.219 Parma_Polyhedra_Library::Linear_System::With_Pending Struct Reference

A tag class.

`#include <Linear_System.defs.hh>`

11.219.1 Detailed Description

A tag class. Tag class to differentiate the Linear_System copy constructor that copies pending rows as pending from the one that transforms pending rows into non-pending ones.

Definition at line 89 of file Linear_System.defs.hh.

The documentation for this struct was generated from the following file:

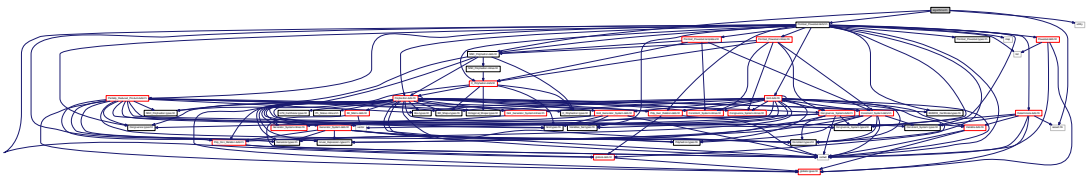
- Linear_System.defs.hh

12 File Documentation

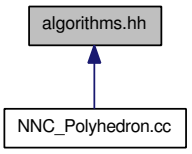
12.1 algorithms.hh File Reference

```
#include "NNC_Polyhedron.defs.hh"
#include "Pointset_Powerset.defs.hh"
#include <utility>
#include "assert.hh"
```

Include dependency graph for algorithms.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

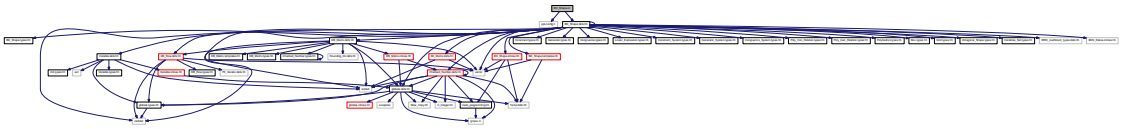
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.2 BD_Shape.cc File Reference

```
#include <ppl-config.h>
#include "BD_Shape.defs.hh"
```

Include dependency graph for BD_Shape.cc:

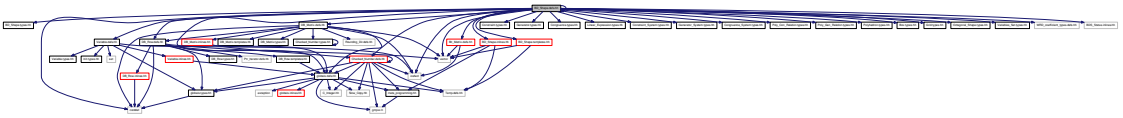


12.3 BD_Shape.defs.hh File Reference

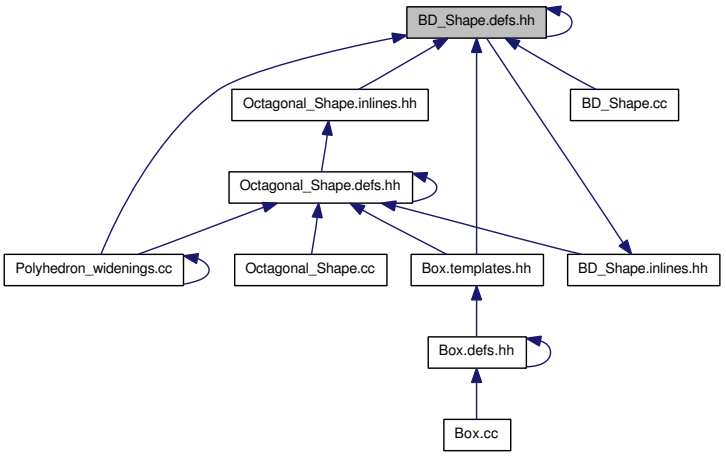
```
#include "BD_Shape.types.hh"
#include "globals.defs.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.types.hh"
#include "Congruence_System.types.hh"
#include "Poly_Con_Relation.types.hh"
#include "Poly_Gen_Relation.types.hh"
#include "Polyhedron.types.hh"
#include "Box.types.hh"
#include "Grid.types.hh"
#include "Octagonal_Shape.types.hh"
```

```
#include "Variable.defs.hh"
#include "Variables_Set.types.hh"
#include "DB_Matrix.defs.hh"
#include "DB_Row.defs.hh"
#include "Checked_Number.defs.hh"
#include "WRD_coefficient_types.defs.hh"
#include "Bit_Matrix.defs.hh"
#include <cstddef>
#include <iosfwd>
#include <vector>
#include "BDS_Status.idefs.hh"
#include "BDS_Status.inlines.hh"
#include "BD_Shape.inlines.hh"
#include "BD_Shape.templates.hh"
```

Include dependency graph for BD_Shape.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::BD_Shape< T >

  A bounded difference shape.

- class Parma_Polyhedra_Library::BD_Shape< T >::Status

A conjunctive assertion about a BD_Shape<T> object.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_IN_BD_Shape_CLASS

12.3.1 Define Documentation

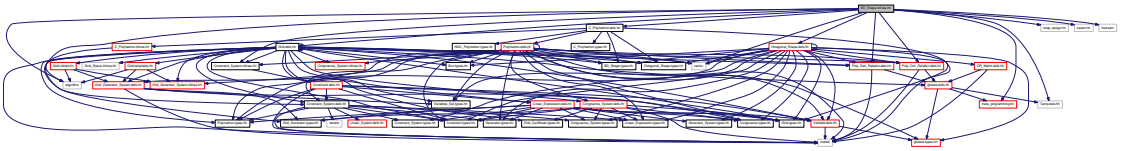
12.3.1.1 #define PPL_IN_BD_Shape_CLASS

Definition at line 1806 of file BD_Shape.defs.hh.

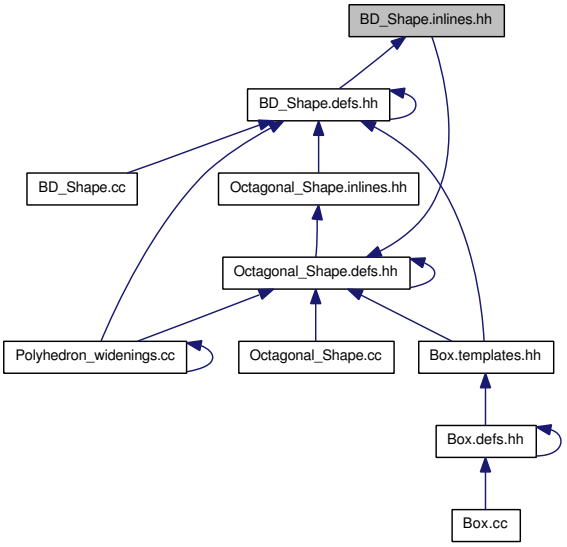
12.4 BD_Shape.inlines.hh File Reference

```
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "C_Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "Octagonal_Shape.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "Temp.defs.hh"
#include "meta_programming.hh"
#include "wrap_assign.hh"
#include "assert.hh"
#include <vector>
#include <iostream>
#include <algorithm>
```

Include dependency graph for BD_Shape.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

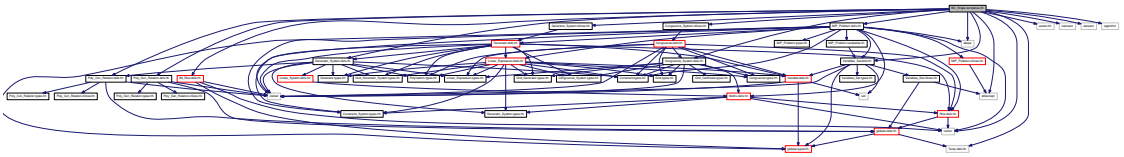
  The standard C++ namespace.

12.5 BD_Shape.templates.hh File Reference

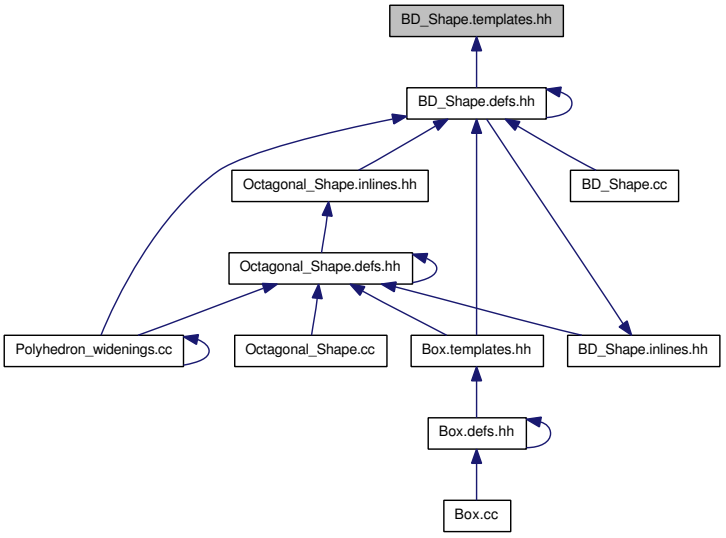
```
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Congruence_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "MIP_Problem.defs.hh"
```

```
#include "Variables_Set.defs.hh"
#include "Bit_Row.defs.hh"
#include "Temp.defs.hh"
#include "assert.hh"
#include <vector>
#include <deque>
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <algorithm>
```

Include dependency graph for BD_Shape.templates.hh:



This graph shows which files directly or indirectly include this file:



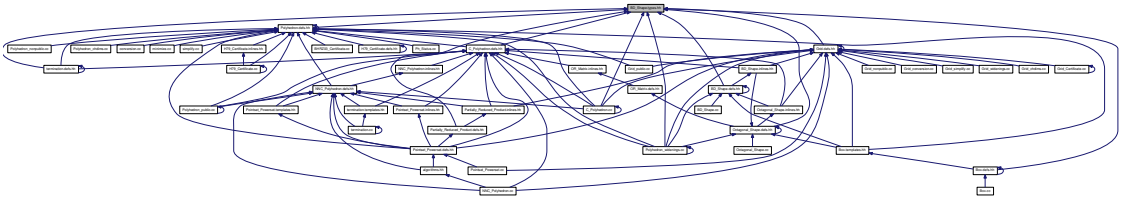
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.6 BD_Shape.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

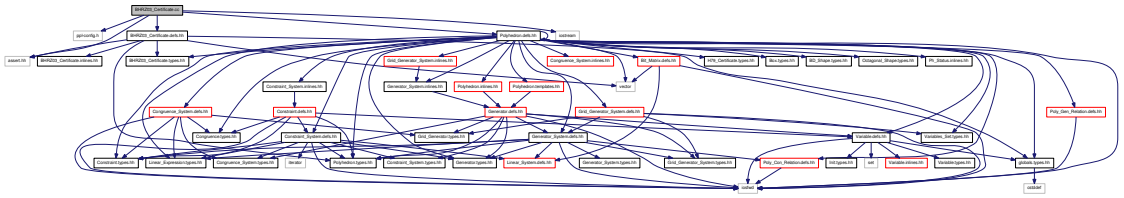
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.7 BHRZ03_Certificate.cc File Reference

```
#include <ppl-config.h>
#include "BHRZ03_Certificate.defs.hh"
#include "Polyhedron.defs.hh"
#include "assert.hh"
#include <iostream>
```

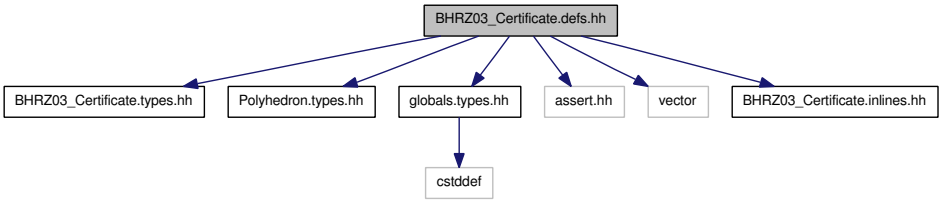
Include dependency graph for BHRZ03_Certificate.cc:



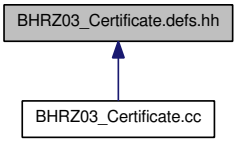
12.8 BHRZ03_Certificate.defs.hh File Reference

```
#include "BHRZ03_Certificate.types.hh"
#include "Polyhedron.types.hh"
#include "globals.types.hh"
#include "assert.hh"
#include <vector>
#include "BHRZ03_Certificate.inlines.hh"
```

Include dependency graph for BHRZ03_Certificate.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::BHRZ03_Certificate

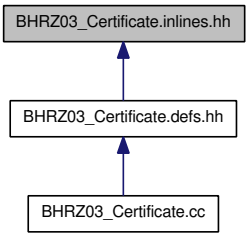
  The convergence certificate for the BHRZ03 widening operator.

- struct Parma_Polyhedra_Library::BHRZ03_Certificate::Compare

  A total ordering on BHRZ03 certificates.

12.9 BHRZ03_Certificate.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



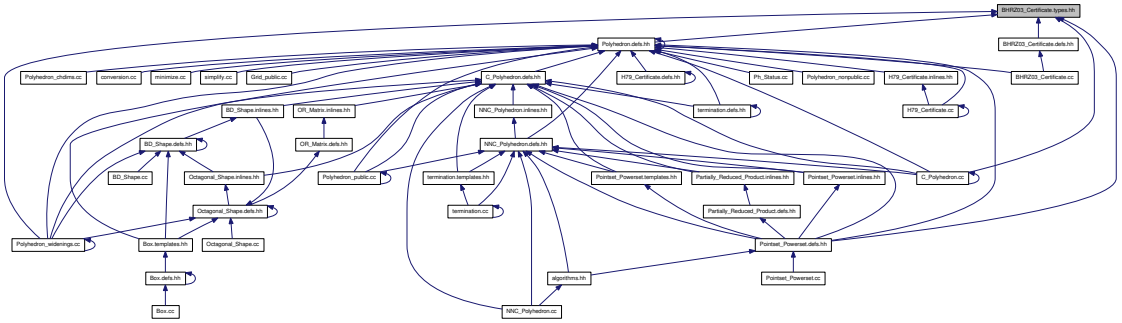
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.10 BHRZ03_Certificate.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

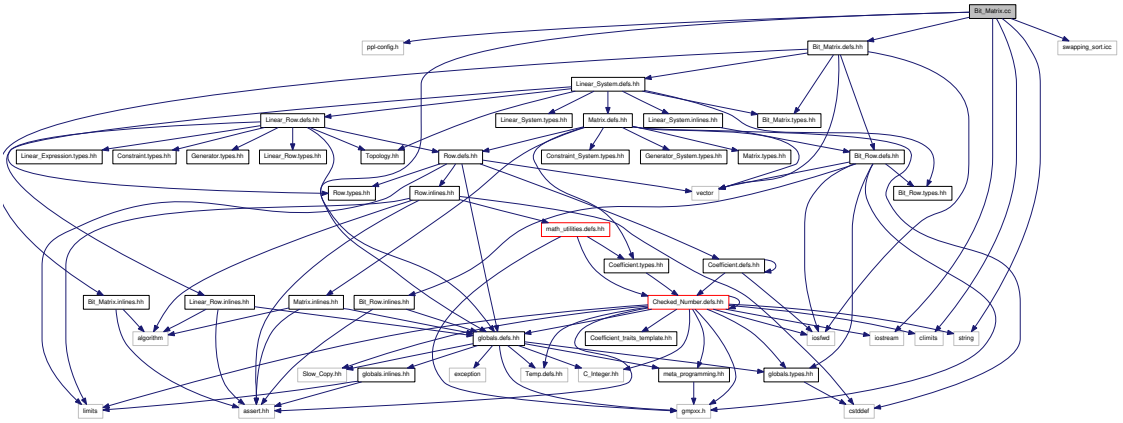
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.11 Bit_Matrix.cc File Reference

```
#include <ppl-config.h>
#include "Bit_Matrix.defs.hh"
#include "globals.defs.hh"
#include <iostream>
#include <string>
#include <climits>
#include "swapping_sort.icc"
```

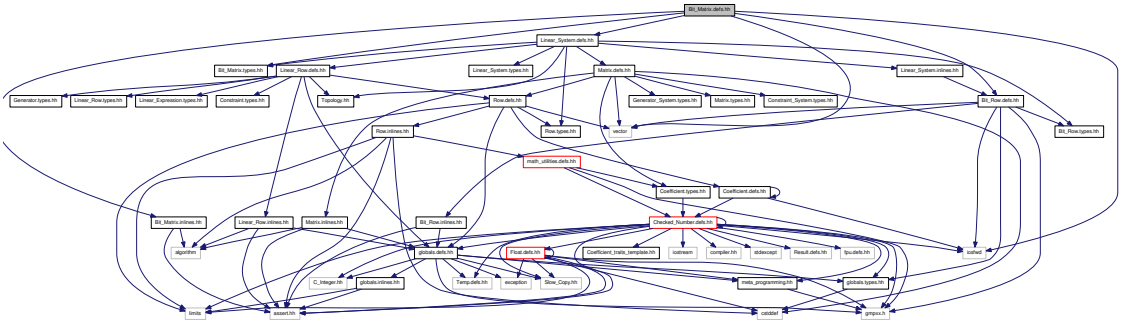
Include dependency graph for Bit_Matrix.cc:



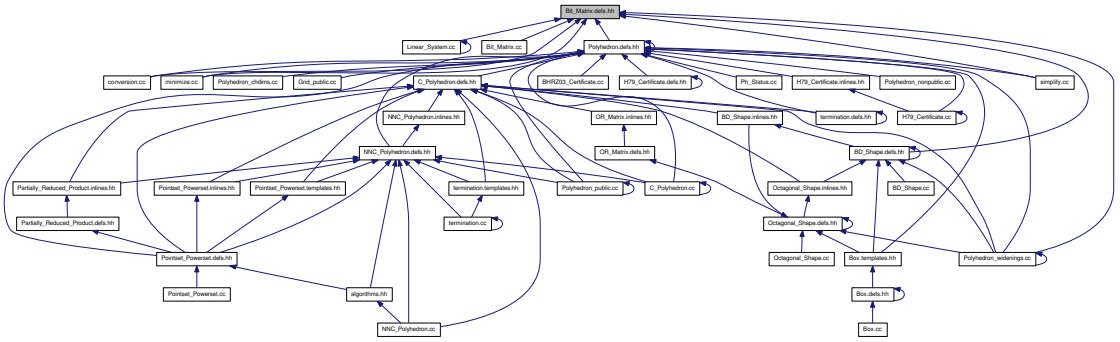
12.12 Bit_Matrix.defs.hh File Reference

```
#include "Bit_Matrix.types.hh"
#include "Linear_System.defs.hh"
#include "Bit_Row.defs.hh"
#include <vector>
#include <iosfwd>
#include "Bit_Matrix.inlines.hh"
```

Include dependency graph for Bit_Matrix.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Bit_Matrix

  A matrix of bits.

- struct Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_Less_Than

  Ordering predicate (used when implementing the sort algorithm).

Namespaces

- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

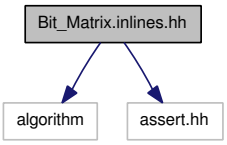
- namespace std

  The standard C++ namespace.

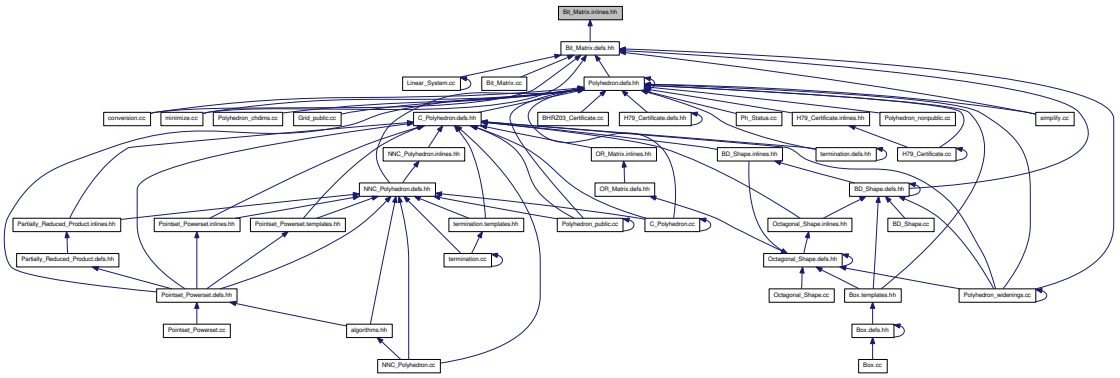
12.13 Bit_Matrix.inlines.hh File Reference

```
#include <algorithm>
#include "assert.hh"
```

Include dependency graph for Bit_Matrix.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

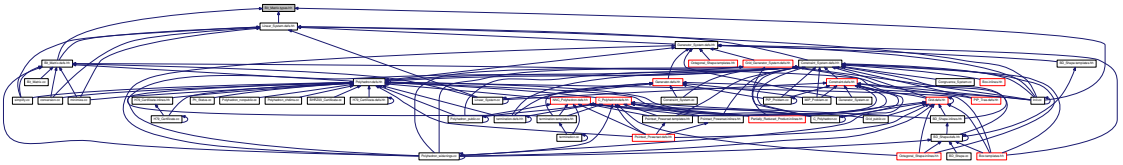
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.14 Bit_Matrix.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

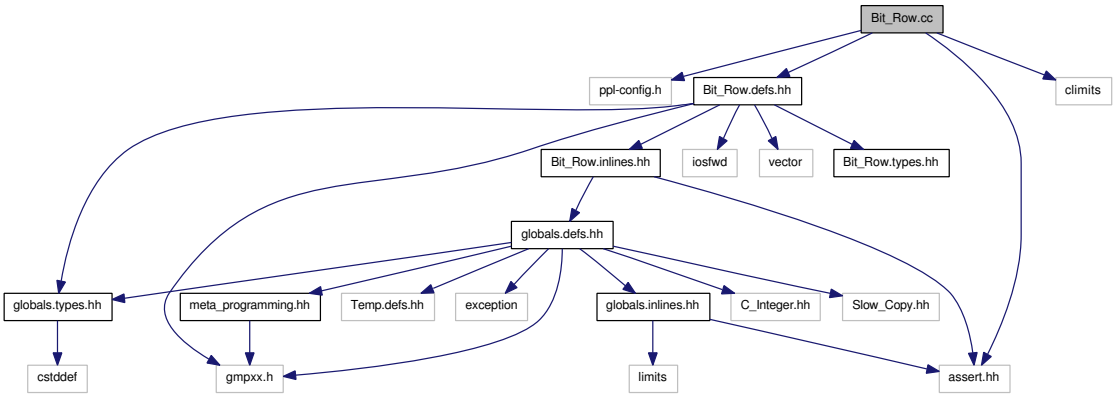
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.15 Bit_Row.cc File Reference

```
#include <ppl-config.h>
#include "Bit_Row.defs.hh"
#include "assert.hh"
#include <climits>
```

Include dependency graph for Bit_Row.cc:

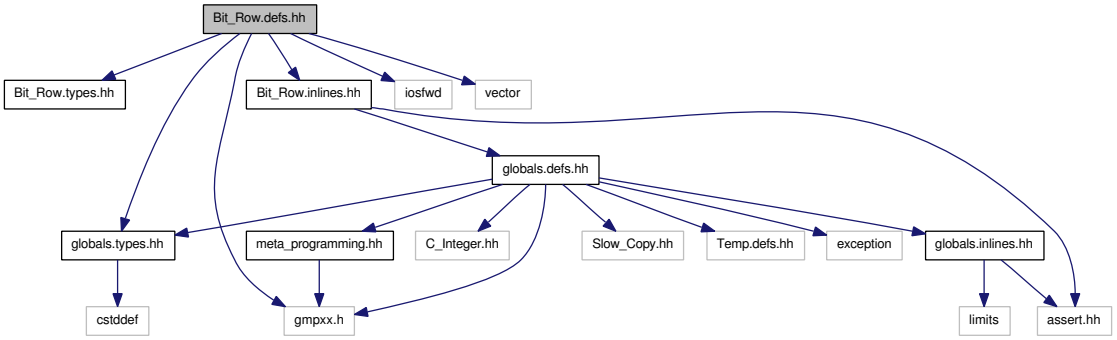


12.16 Bit_Row.defs.hh File Reference

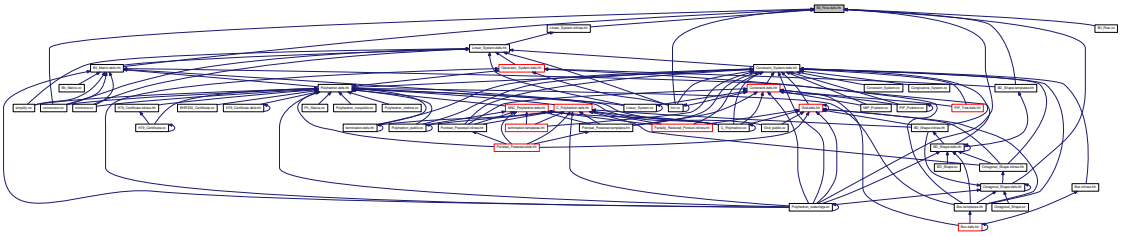
```
#include "Bit_Row.types.hh"
#include "globals.types.hh"
#include <iosfwd>
#include <gmpxx.h>
#include <vector>
```

```
#include "Bit_Row.inlines.hh"
```

Include dependency graph for Bit_Row.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Bit_Row

  A row in a matrix of bits.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

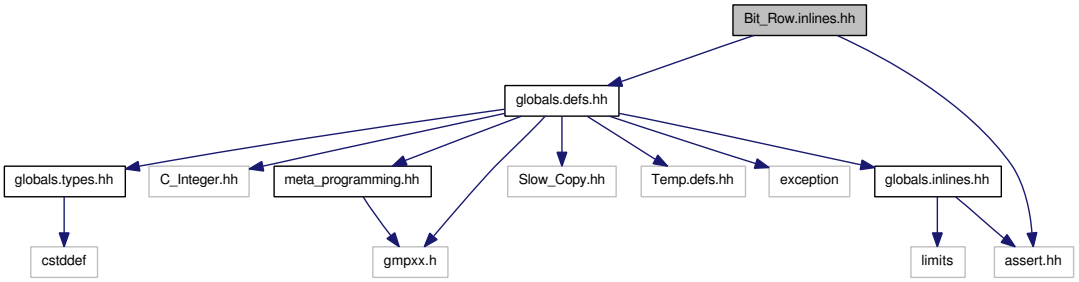
- namespace std

  The standard C++ namespace.

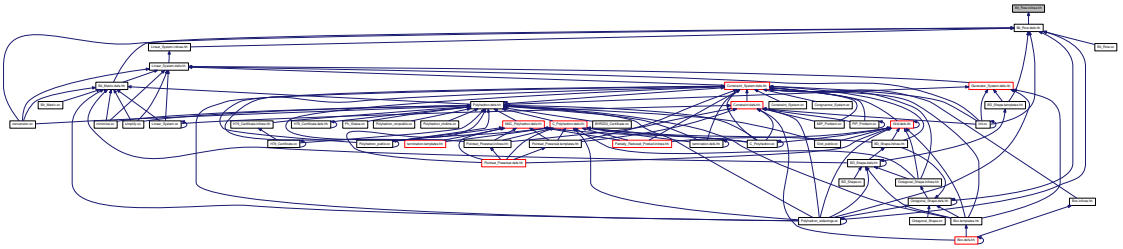
12.17 Bit_Row.inlines.hh File Reference

```
#include "globals.defs.hh"
#include "assert.hh"
```

Include dependency graph for Bit_Row.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_BITS_PER_GMP_LIMB (PPL_SIZEOF_MP_LIMB_T∗CHAR_BIT)

Functions

- unsigned int Parma_Polyhedra_Library::Implementation::first_one (mp_limb_t w)

  Assuming `w` *is nonzero, returns the index of the first set bit in* `w`*.*

- unsigned int Parma_Polyhedra_Library::Implementation::last_one (mp_limb_t w)

  Assuming `w` *is nonzero, returns the index of the last set bit in* `w`*.*

12.17.1 Define Documentation

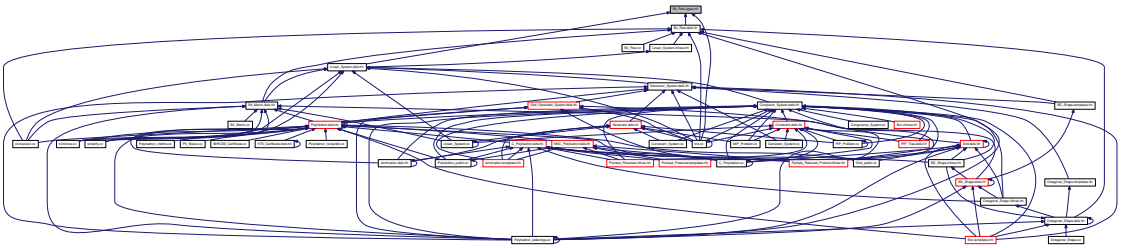
12.17.1.1 #define PPL_BITS_PER_GMP_LIMB (PPL_SIZEOF_MP_LIMB_T∗CHAR_BIT)

Definition at line 37 of file Bit_Row.inlines.hh.

Referenced by Parma_Polyhedra_Library::Bit_Row::Bit_Row(), Parma_Polyhedra_Library::Bit_-Row::first(), Parma_Polyhedra_Library::Bit_Row::last(), Parma_Polyhedra_Library::Bit_Row::next(), and Parma_Polyhedra_Library::Bit_Row::prev().

12.18 Bit_Row.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

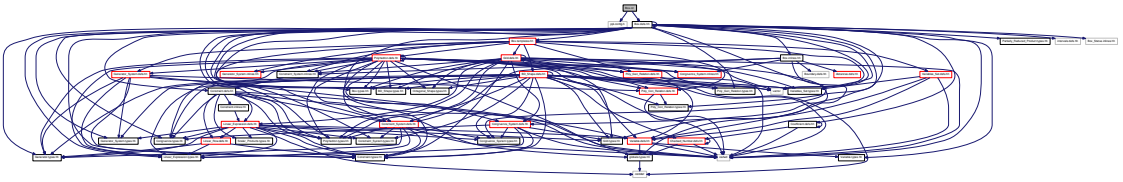
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.19 Box.cc File Reference

```
#include <ppl-config.h>
#include "Box.defs.hh"
```

Include dependency graph for Box.cc:

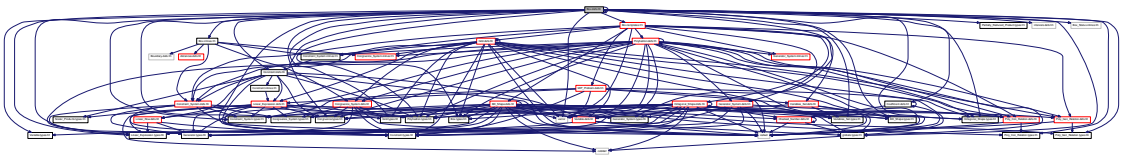


12.20 Box.defs.hh File Reference

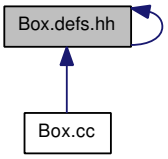
```
#include "Box.types.hh"
#include "globals.types.hh"
```

```
#include "Coefficient.defs.hh"
#include "Variable.types.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Constraint.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator.types.hh"
#include "Generator_System.types.hh"
#include "Congruence.types.hh"
#include "Congruence_System.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include "Poly_Con_Relation.types.hh"
#include "Poly_Gen_Relation.types.hh"
#include "Polyhedron.types.hh"
#include "Grid.types.hh"
#include "Partially_Reduced_Product.types.hh"
#include "intervals.defs.hh"
#include <vector>
#include <iosfwd>
#include "Box_Status.idefs.hh"
#include "Box_Status.inlines.hh"
#include "Box.inlines.hh"
#include "Box.templates.hh"
```

Include dependency graph for Box.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Box< ITV >

  A not necessarily closed, iso-oriented hyperrectangle.

- class Parma_Polyhedra_Library::Box< ITV >::Status

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

Defines

- #define PPL_IN_Box_CLASS

Functions

- template<typename ITV >
  Poly_Con_Relation Parma_Polyhedra_Library::interval_relation (const ITV &i, const Constraint::Type constraint_type, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den=1)

  Returns the relations holding between an interval and an interval constraint.

- bool Parma_Polyhedra_Library::extract_interval_congruence (const Congruence &cg, dimension_type cg_space_dim, dimension_type &cg_num_vars, dimension_type &cg_only_var)

12.20.1 Define Documentation

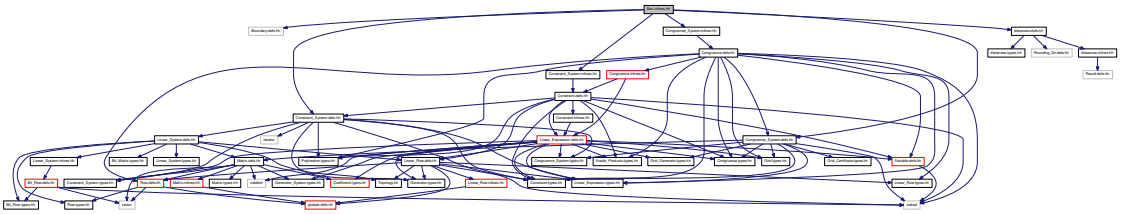
12.20.1.1 #define PPL_IN_Box_CLASS

Definition at line 1704 of file Box.defs.hh.

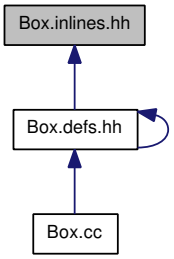
12.21 Box.inlines.hh File Reference

```
#include "Boundary.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "distances.defs.hh"
```

Include dependency graph for Box.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

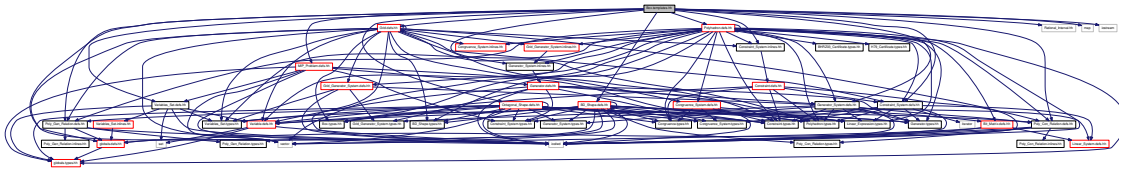
  The entire library is confined to this namespace.

12.22 Box.templates.hh File Reference

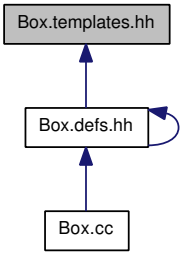
```
#include "Variables_Set.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "BD_Shape.defs.hh"
#include "Octagonal_Shape.defs.hh"
#include "MIP_Problem.defs.hh"
#include "Rational_Interval.hh"
#include <vector>
```

```
#include <map>
#include <iostream>
```

Include dependency graph for Box.templates.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

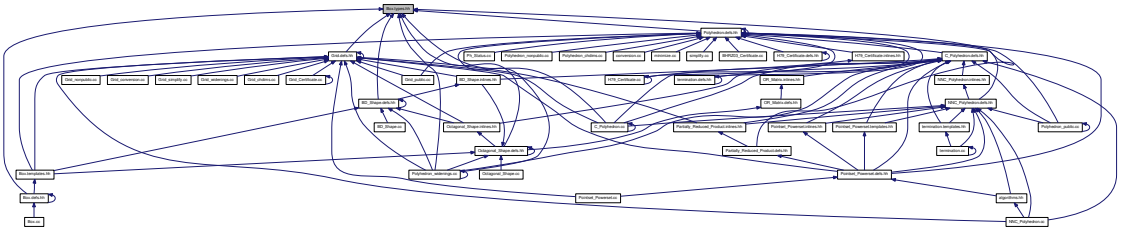
- template<typename ITV >
  bool Parma_Polyhedra_Library::operator== (const Box< ITV > &x, const Box< ITV > &y)

- template<typename ITV >
  Poly_Con_Relation Parma_Polyhedra_Library::interval_relation (const ITV &i, const Constraint::Type constraint_type, Coefficient_traits::const_reference num, Coefficient_traits::const_reference den=1)

  Returns the relations holding between an interval and an interval constraint.

12.23 Box.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

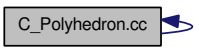
  The entire library is confined to this namespace.

12.24 C_Polyhedron.cc File Reference

```
#include <ppl-config.h>
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "algorithms.hh"
#include "globals.defs.hh"
#include "BHRZ03_Certificate.types.hh"
#include "Constraint.types.hh"
#include "Constraint_System.types.hh"
#include "Congruence.types.hh"
#include "Congruence_System.types.hh"
#include "Polyhedron.defs.hh"
#include "Partially_Reduced_Product.types.hh"
#include "globals.types.hh"
#include "Coefficient.defs.hh"
#include "Variable.types.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.types.hh"
#include "Generator.types.hh"
#include "Grid_Generator.types.hh"
#include "Generator_System.types.hh"
```

```
#include "Grid_Generator_System.types.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "C_Polyhedron.types.hh"
#include "NNC_Polyhedron.types.hh"
#include "Grid.types.hh"
#include "Box.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include "Constraint_System.defs.hh"
#include "Congruence_System.defs.hh"
#include "Grid_Generator.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include <algorithm>
#include <deque>
#include "Variable.defs.hh"
#include <iosfwd>
#include "assert.hh"
#include "Powerset.types.hh"
#include <iterator>
#include <list>
#include <iostream>
#include <map>
#include "Constraint.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence.defs.hh"
#include "Congruence_System.inlines.hh"
#include "Variables_Set.defs.hh"
#include <string>
#include <sstream>
#include <stdexcept>
#include <utility>
```

This graph shows which files directly or indirectly include this file:



Defines

- #define USE_BHZ09 0
- #define USE_BFT00 1

12.24.1 Define Documentation

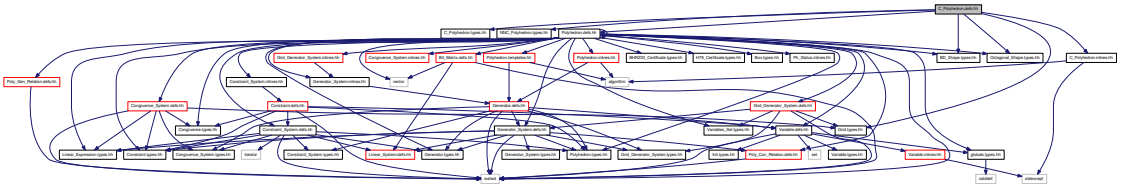
12.24.1.1 #define USE_BFT00 1

12.24.1.2 #define USE_BHZ09 0

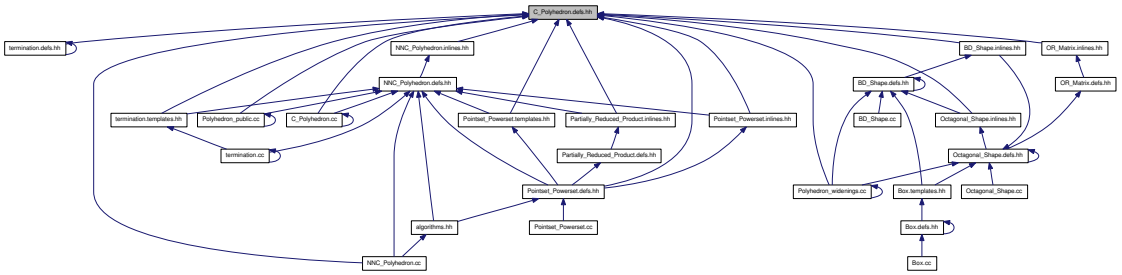
12.25 C_Polyhedron.defs.hh File Reference

```
#include "C_Polyhedron.types.hh"
#include "NNC_Polyhedron.types.hh"
#include "Polyhedron.defs.hh"
#include "Grid.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include "C_Polyhedron.inlines.hh"
```

Include dependency graph for C_Polyhedron.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

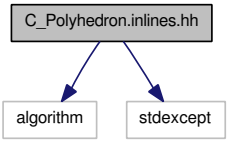
- class Parma_Polyhedra_Library::C_Polyhedron

  A closed convex polyhedron.

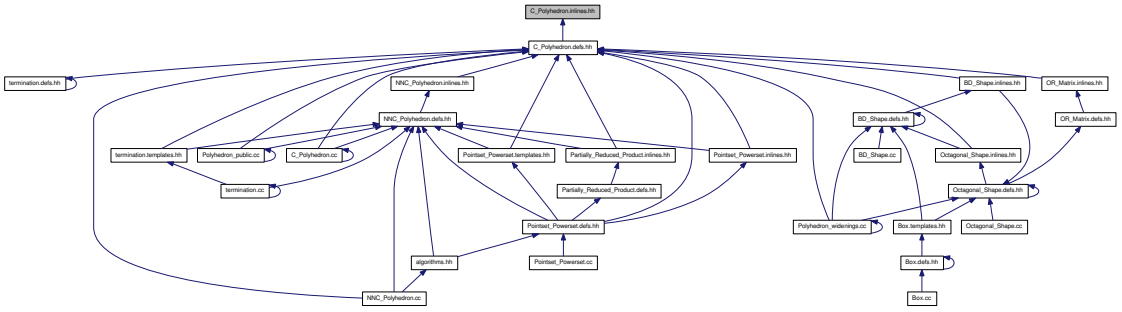
12.26 C_Polyhedron.inlines.hh File Reference

```
#include <algorithm>
#include <stdexcept>
```

Include dependency graph for C_Polyhedron.inlines.hh:



This graph shows which files directly or indirectly include this file:



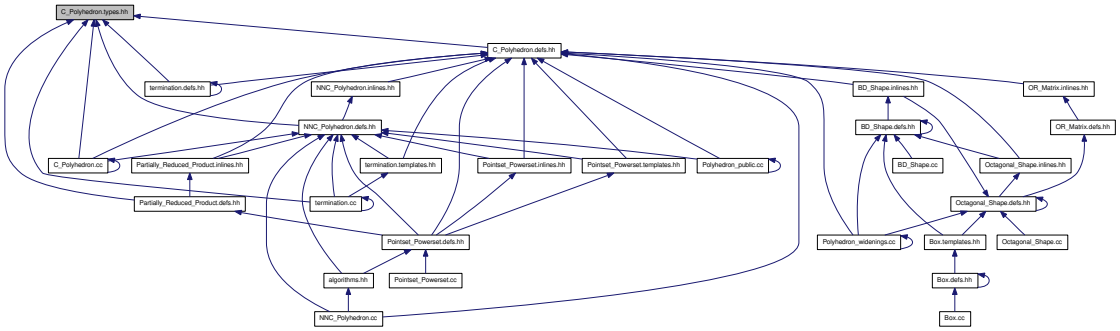
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.27 C_Polyhedron.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

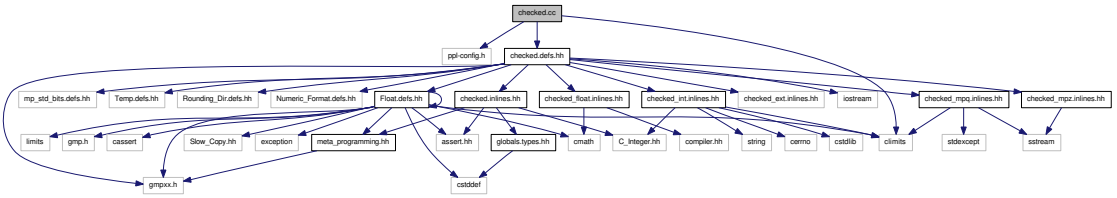
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.28 checked.cc File Reference

```
#include <ppl-config.h>
#include "checked.defs.hh"
#include <climits>
```

Include dependency graph for checked.cc:



Classes

- struct Parma_Polyhedra_Library::Checked::number_struct

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Functions

- int Parma_Polyhedra_Library::Checked::get_digit (int c, int base=10)

  Returns the integer value associated with the ASCII code `c`*, in the base* `base` *positional number system, if there is such an association; returns* -1 *otherwise.*

- bool Parma_Polyhedra_Library::Checked::sum_sign (bool &a_neg, unsigned long &a_mod, bool b_neg, unsigned long b_mod)

  Adds the number represented (in the modulus-and-sign representation) by `b_neg` *and* `b_mod` *to the number represented by* `a_neg` *and* `a_mod`*, assigning the result to the latter. Returns* `false` *is the result cannot be represented; returns* `true` *otherwise.*

- Result Parma_Polyhedra_Library::Checked::parse_number_part (std::istream &is, number_struct &num)

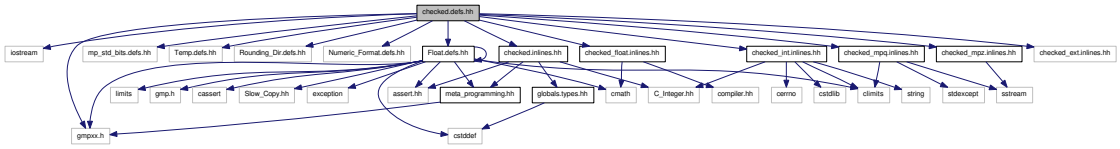
  Helper function for parse_number(): reads the numerator or denominator part of a number from `is` *into* `num`*, returning the appropriate Result value.*

- Result Parma_Polyhedra_Library::Checked::parse_number (std::istream &is, number_struct &num, number_struct &den)
- Result Parma_Polyhedra_Library::Checked::input_mpq (mpq_class &to, std::istream &is)

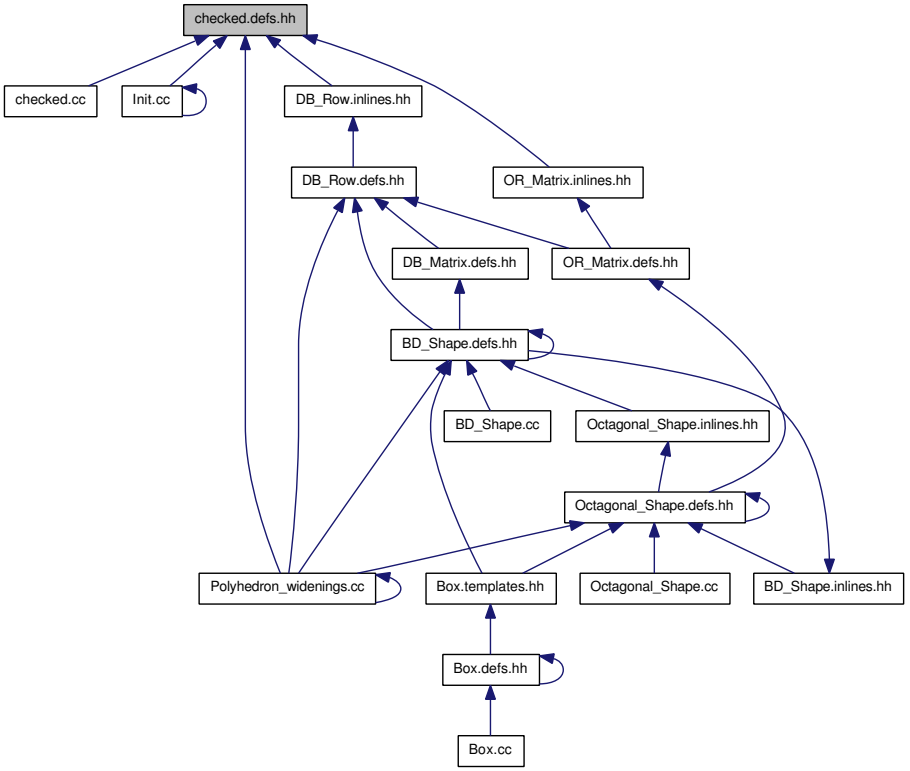
12.29 checked.defs.hh File Reference

```
#include <iostream>
#include <gmpxx.h>
#include "mp_std_bits.defs.hh"
#include "Temp.defs.hh"
#include "Rounding_Dir.defs.hh"
#include "Numeric_Format.defs.hh"
#include "Float.defs.hh"
#include "checked.inlines.hh"
#include "checked_int.inlines.hh"
#include "checked_float.inlines.hh"
#include "checked_mpz.inlines.hh"
#include "checked_mpq.inlines.hh"
#include "checked_ext.inlines.hh"
```

Include dependency graph for checked.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Minus_Infinity
- struct Parma_Polyhedra_Library::Plus_Infinity
- struct Parma_Polyhedra_Library::Not_A_Number
- struct Parma_Polyhedra_Library::Is_Special< T >
- struct Parma_Polyhedra_Library::Is_Special< Minus_Infinity >
- struct Parma_Polyhedra_Library::Is_Special< Plus_Infinity >
- struct Parma_Polyhedra_Library::Is_Special< Not_A_Number >
- struct Parma_Polyhedra_Library::Checked_Number_Transparent_Policy< T >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Defines

- #define PPL_FUNCTION_CLASS(name) name ## _function_struct

- #define PPL_DECLARE_FUN1_0_0(name, ret_type, qual, type)
- #define PPL_DECLARE_FUN1_0_1(name, ret_type, qual, type, after1)
- #define PPL_DECLARE_FUN1_0_2(name, ret_type, qual, type, after1, after2)
- #define PPL_DECLARE_FUN1_0_3(name, ret_type, qual, type,after1, after2, after3)
- #define PPL_DECLARE_FUN1_1_1(name, ret_type, before1, qual, type, after1)
- #define PPL_DECLARE_FUN1_1_2(name, ret_type, before1, qual, type,after1, after2)
- #define PPL_DECLARE_FUN1_2_2(name, ret_type, before1, before2, qual, type, after1, after2)
- #define PPL_DECLARE_FUN2_0_0(name, ret_type, qual1, type1, qual2, type2)
- #define PPL_DECLARE_FUN2_0_1(name, ret_type, qual1, type1,qual2, type2, after1)
- #define PPL_DECLARE_FUN2_0_2(name, ret_type, qual1, type1, qual2, type2, after1, after2)
- #define PPL_DECLARE_FUN3_0_1(name, ret_type, qual1, type1,qual2, type2, qual3, type3, after1)
- #define PPL_DECLARE_FUN5_0_1(name, ret_type,qual1, type1, qual2, type2, qual3, type3, qual4, type4, qual5, type5,after1)
- #define PPL_SPECIALIZE_FUN1_0_0(name, func, ret_type, qual, type)
- #define PPL_SPECIALIZE_FUN1_0_1(name, func, ret_type, qual, type, after1)
- #define PPL_SPECIALIZE_FUN1_0_2(name, func, ret_type, qual, type,after1, after2)
- #define PPL_SPECIALIZE_FUN1_0_3(name, func, ret_type, qual, type,after1, after2, after3)
- #define PPL_SPECIALIZE_FUN1_1_1(name, func, ret_type, before1,qual, type, after1)
- #define PPL_SPECIALIZE_FUN1_1_2(name, func, ret_type, before1,qual, type, after1, after2)
- #define PPL_SPECIALIZE_FUN1_2_2(name, func, ret_type, before1, before2, qual, type, after1, after2)
- #define PPL_SPECIALIZE_FUN2_0_0(name, func, ret_type, qual1, type1,qual2, type2)
- #define PPL_SPECIALIZE_FUN2_0_1(name, func, ret_type, qual1, type1,qual2, type2, after1)
- #define PPL_SPECIALIZE_FUN2_0_2(name, func, ret_type, qual1, type1,qual2, type2, after1, after2)
- #define PPL_SPECIALIZE_FUN3_0_1(name, func, ret_type, qual1, type1,qual2, type2, qual3, type3, after1)
- #define PPL_SPECIALIZE_FUN5_0_1(name, func, ret_type,qual1, type1, qual2, type2,qual3, type3,qual4, type4, qual5, type5, after1)
- #define nonconst
- #define PPL_SPECIALIZE_COPY(func, Type) PPL_SPECIALIZE_FUN2_0_0(copy, func, void, nonconst, Type, const, Type)
- #define PPL_SPECIALIZE_SGN(func, From) PPL_SPECIALIZE_FUN1_0_0(sgn, func, Result_-Relation, const, From)
- #define PPL_SPECIALIZE_CMP(func, Type1, Type2) PPL_SPECIALIZE_FUN2_0_0(cmp, func, Result_Relation, const, Type1, const, Type2)
- #define PPL_SPECIALIZE_CLASSIFY(func, Type) PPL_SPECIALIZE_FUN1_0_3(classify, func, Result, const, Type, bool, bool, bool)
- #define PPL_SPECIALIZE_IS_NAN(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_nan, func, bool, const, Type)
- #define PPL_SPECIALIZE_IS_MINF(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_minf, func, bool, const, Type)
- #define PPL_SPECIALIZE_IS_PINF(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_pinf, func, bool, const, Type)
- #define PPL_SPECIALIZE_IS_INT(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_int, func, bool, const, Type)
- #define PPL_SPECIALIZE_ASSIGN_SPECIAL(func, Type)
- #define PPL_SPECIALIZE_CONSTRUCT_SPECIAL(func, Type)
- #define PPL_SPECIALIZE_CONSTRUCT(func, To, From)
- #define PPL_SPECIALIZE_ASSIGN(func, To, From)

- #define PPL_SPECIALIZE_FLOOR(func, To, From)
- #define PPL_SPECIALIZE_CEIL(func, To, From)
- #define PPL_SPECIALIZE_TRUNC(func, To, From)
- #define PPL_SPECIALIZE_NEG(func, To, From)
- #define PPL_SPECIALIZE_ABS(func, To, From)
- #define PPL_SPECIALIZE_SQRT(func, To, From)
- #define PPL_SPECIALIZE_ADD(func, To, From1, From2)
- #define PPL_SPECIALIZE_SUB(func, To, From1, From2)
- #define PPL_SPECIALIZE_MUL(func, To, From1, From2)
- #define PPL_SPECIALIZE_DIV(func, To, From1, From2)
- #define PPL_SPECIALIZE_REM(func, To, From1, From2)
- #define PPL_SPECIALIZE_IDIV(func, To, From1, From2)
- #define PPL_SPECIALIZE_ADD_2EXP(func, To, From)
- #define PPL_SPECIALIZE_SUB_2EXP(func, To, From)
- #define PPL_SPECIALIZE_MUL_2EXP(func, To, From)
- #define PPL_SPECIALIZE_DIV_2EXP(func, To, From)
- #define PPL_SPECIALIZE_SMOD_2EXP(func, To, From)
- #define PPL_SPECIALIZE_UMOD_2EXP(func, To, From)
- #define PPL_SPECIALIZE_ADD_MUL(func, To, From1, From2)
- #define PPL_SPECIALIZE_SUB_MUL(func, To, From1, From2)
- #define PPL_SPECIALIZE_GCD(func, To, From1, From2)
- #define PPL_SPECIALIZE_GCDEXT(func, To1, From1, From2, To2, To3)
- #define PPL_SPECIALIZE_LCM(func, To, From1, From2)
- #define PPL_SPECIALIZE_INPUT(func, Type)
- #define PPL_SPECIALIZE_OUTPUT(func, Type)
- #define CHECK_P(cond, check) ((cond) ? (check) : (assert(!(check)), false))

Functions

- Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_0 (copy, void, nonconst, Type1, const, Type2) PPL_DECLARE_FUN1_0_0(sgn
- From Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_0 (cmp, Result_Relation, const, Type1, const, Type2) PPL_DECLARE_FUN1_0_3(classify
- From bool Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_0_0 (is_nan, bool, const, Type) PPL_DECLARE_FUN1_0_0(is_minf
- From bool Type Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_0_0 (is_pinf, bool, const, Type) PPL_DECLARE_FUN1_0_0(is_int
- From bool Type Type Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_0_2 (assign_-special, Result, nonconst, Type, Result_Class, Rounding_Dir) PPL_DECLARE_FUN1_0_-2(construct_special
- From bool Type Type Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_-FUN2_0_1 (construct, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_-0_1(assign
- From bool Type Type Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_-DECLARE_FUN2_0_1 (floor, Result, nonconst, To, const, From, Rounding_Dir) PPL_-DECLARE_FUN2_0_1(ceil
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_-Library::Checked::PPL_DECLARE_FUN2_0_1 (trunc, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_0_1(neg

- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_1 (abs, Result, nonconst, To, const, From, Rounding_Dir) PPL_DECLARE_FUN2_0_1(sqrt
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (add, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(sub
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(div
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (rem, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(idiv
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (add_2exp, Result, nonconst, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_FUN2_0_2(sub_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (mul_2exp, Result, nonconst, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_FUN2_0_2(div_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN2_0_2 (smod_2exp, Result, nonconst, To, const, From, unsigned int, Rounding_Dir) PPL_DECLARE_FUN2_0_2(umod_2exp
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (add_mul, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN3_0_1(sub_mul
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (gcd, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN5_0_1(gcdext
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN3_0_1 (lcm, Result, nonconst, To, const, From1, const, From2, Rounding_Dir) PPL_DECLARE_FUN1_0_2(input
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir Parma_Polyhedra_Library::Checked::PPL_DECLARE_FUN1_1_2 (output, Result, std::ostream &, const, Type, const Numeric_Format &, Rounding_Dir) template< typename Policy
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Rounding_Dir std::istream Rounding_Dir To Result Parma_Polyhedra_Library::Checked::round (To &to, Result r, Rounding_Dir dir)
- Result Parma_Polyhedra_Library::Checked::input_mpq (mpq_class &to, std::istream &is)

Variables

- Parma_Polyhedra_Library::Checked::Result_Relation
- Parma_Polyhedra_Library::Checked::const
- From Parma_Polyhedra_Library::Checked::Result
- From Parma_Polyhedra_Library::Checked::Type
- From Parma_Polyhedra_Library::Checked::bool
- From bool Type Type Parma_Polyhedra_Library::Checked::nonconst
- From bool Type Type Parma_Polyhedra_Library::Checked::Result_Class
- From bool Type Type Rounding_Dir Parma_Polyhedra_Library::Checked::To
- From bool Type Type Rounding_Dir Parma_Polyhedra_Library::Checked::From
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::From1
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::From2
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Parma_Polyhedra_Library::Checked::int
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To1
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To2
- From bool Type Type Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir unsigned Rounding_Dir Rounding_Dir Parma_Polyhedra_Library::Checked::To3
- Minus_Infinity Parma_Polyhedra_Library::MINUS_INFINITY
- Plus_Infinity Parma_Polyhedra_Library::PLUS_INFINITY
- Not_A_Number Parma_Polyhedra_Library::NOT_A_NUMBER

12.29.1 Define Documentation

12.29.1.1 #define CHECK_P(cond, check) ((cond) ? (check) : (assert(!(check)), false))

Definition at line 652 of file checked.defs.hh.

Referenced by Parma_Polyhedra_Library::Checked::add_unsigned_int(), Parma_Polyhedra_Library::Checked::assign_int_float(), Parma_Polyhedra_Library::Checked::assign_signed_int_signed_int(), Parma_Polyhedra_Library::Checked::assign_signed_int_unsigned_int(), Parma_Polyhedra_Library::Checked::assign_unsigned_int_mpz(), Parma_Polyhedra_Library::Checked::assign_unsigned_int_signed_int(), Parma_Polyhedra_Library::Checked::assign_unsigned_int_unsigned_int(), Parma_Polyhedra_Library::Checked::classify_float(), Parma_Polyhedra_Library::Checked::div_mpq(), Parma_Polyhedra_Library::Checked::div_mpz(), Parma_Polyhedra_Library::Checked::div_signed_int(), Parma_Polyhedra_Library::Checked::div_unsigned_int(), Parma_Polyhedra_Library::Checked::idiv_mpq(), Parma_Polyhedra_Library::Checked::idiv_mpz(), Parma_Polyhedra_Library::Checked::idiv_signed_int(), Parma_Polyhedra_Library::Checked::idiv_unsigned_int(), Parma_Polyhedra_Library::Checked::is_inf_float(), Parma_Polyhedra_Library::Checked::is_nan_float(), Parma_Polyhedra_Library::Checked::neg_signed_int(), Parma_Polyhedra_Library::Checked::neg_unsigned_int(), Parma_Polyhedra_Library::Checked::rem_mpq(), Parma_Polyhedra_Library::Checked::rem_mpz(), Parma_Polyhedra_Library::Checked::rem_signed_int(), Parma_Polyhedra_Library::Checked::rem_unsigned_int(), Parma_Polyhedra_Library::Checked::sqrt_mpq(), Parma_Polyhedra_Library::Checked::sqrt_mpz(),

Parma_Polyhedra_Library::Checked::sqrt_signed_int(), and Parma_Polyhedra_Library::Checked::sub_-unsigned_int().

12.29.1.2 #define nonconst

Definition at line 318 of file checked.defs.hh.

12.29.1.3 #define PPL_DECLARE_FUN1_0_0(name, ret_type, qual, type)

Definition at line 47 of file checked.defs.hh.

12.29.1.4 #define PPL_DECLARE_FUN1_0_1(name, ret_type, qual, type, after1)

Definition at line 55 of file checked.defs.hh.

12.29.1.5 #define PPL_DECLARE_FUN1_0_2(name, ret_type, qual, type, after1, after2)

Definition at line 63 of file checked.defs.hh.

12.29.1.6 #define PPL_DECLARE_FUN1_0_3(name, ret_type, qual, type, after1, after2, after3)

Definition at line 72 of file checked.defs.hh.

12.29.1.7 #define PPL_DECLARE_FUN1_1_1(name, ret_type, before1, qual, type, after1)

Definition at line 82 of file checked.defs.hh.

12.29.1.8 #define PPL_DECLARE_FUN1_1_2(name, ret_type, before1, qual, type, after1, after2)

Definition at line 91 of file checked.defs.hh.

12.29.1.9 #define PPL_DECLARE_FUN1_2_2(name, ret_type, before1, before2, qual, type, after1, after2)

Definition at line 102 of file checked.defs.hh.

12.29.1.10 #define PPL_DECLARE_FUN2_0_0(name, ret_type, qual1, type1, qual2, type2)

Definition at line 112 of file checked.defs.hh.

12.29.1.11 #define PPL_DECLARE_FUN2_0_1(name, ret_type, qual1, type1, qual2, type2, after1)

Definition at line 124 of file checked.defs.hh.

12.29.1.12 #define PPL_DECLARE_FUN2_0_2(name, ret_type, qual1, type1, qual2, type2, after1, after2)

Definition at line 137 of file checked.defs.hh.

12.29.1.13 #define PPL_DECLARE_FUN3_0_1(name, ret_type, qual1, type1, qual2, type2, qual3, type3, after1)

Definition at line 150 of file checked.defs.hh.

12.29.1.14 #define PPL_DECLARE_FUN5_0_1(name, ret_type, qual1, type1, qual2, type2, qual3, type3, qual4, type4, qual5, type5, after1)

Definition at line 165 of file checked.defs.hh.

12.29.1.15 #define PPL_FUNCTION_CLASS(name) name ## _function_struct

Definition at line 45 of file checked.defs.hh.

12.29.1.16 #define PPL_SPECIALIZE_ABS(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(abs, func, Result, nonconst, To,              \
                        const, From, Rounding_Dir)
```

Definition at line 360 of file checked.defs.hh.

12.29.1.17 #define PPL_SPECIALIZE_ADD(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(add, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 366 of file checked.defs.hh.

12.29.1.18 #define PPL_SPECIALIZE_ADD_2EXP(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_2(add_2exp, func, Result, nonconst, To,         \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 384 of file checked.defs.hh.

12.29.1.19 #define PPL_SPECIALIZE_ADD_MUL(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(add_mul, func, Result, nonconst, To,          \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 402 of file checked.defs.hh.

12.29.1.20 #define PPL_SPECIALIZE_ASSIGN(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(assign, func, Result, nonconst, To,           \
                        const, From, Rounding_Dir)
```

Definition at line 345 of file checked.defs.hh.

12.29.1.21 #define PPL_SPECIALIZE_ASSIGN_SPECIAL(func, Type)

Value:

```
PPL_SPECIALIZE_FUN1_0_2(assign_special, func, Result,                 \
                        nonconst, Type, Result_Class, Rounding_Dir)
```

Definition at line 336 of file checked.defs.hh.

12.29.1.22 #define PPL_SPECIALIZE_CEIL(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(ceil, func, Result, nonconst, To, \
                        const, From, Rounding_Dir)
```

Definition at line 351 of file checked.defs.hh.

12.29.1.23 #define PPL_SPECIALIZE_CLASSIFY(func, Type) PPL_SPECIALIZE_FUN1_0_3(classify, func, Result, const, Type, bool, bool, bool)

Definition at line 326 of file checked.defs.hh.

12.29.1.24 #define PPL_SPECIALIZE_CMP(func, Type1, Type2) PPL_SPECIALIZE_FUN2_0_0(cmp, func, Result_Relation, const, Type1, const, Type2)

Definition at line 324 of file checked.defs.hh.

12.29.1.25 #define PPL_SPECIALIZE_CONSTRUCT(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(construct, func, Result, nonconst, To, \
                        const, From, Rounding_Dir)
```

Definition at line 342 of file checked.defs.hh.

12.29.1.26 #define PPL_SPECIALIZE_CONSTRUCT_SPECIAL(func, Type)

Value:

```
PPL_SPECIALIZE_FUN1_0_2(construct_special, func, Result, nonconst,    \
                        Type, Result_Class, Rounding_Dir)
```

Definition at line 339 of file checked.defs.hh.

12.29.1.27 #define PPL_SPECIALIZE_COPY(func, Type) PPL_SPECIALIZE_FUN2_0_0(copy, func, void, nonconst, Type, const, Type)

Definition at line 320 of file checked.defs.hh.

12.29.1.28 #define PPL_SPECIALIZE_DIV(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(div, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 375 of file checked.defs.hh.

12.29.1.29 #define PPL_SPECIALIZE_DIV_2EXP(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_2(div_2exp, func, Result, nonconst, To,         \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 393 of file checked.defs.hh.

12.29.1.30 #define PPL_SPECIALIZE_FLOOR(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(floor, func, Result, nonconst, To,            \
                        const, From, Rounding_Dir)
```

Definition at line 348 of file checked.defs.hh.

12.29.1.31 #define PPL_SPECIALIZE_FUN1_0_0(name, func, ret_type, qual, type)

Definition at line 183 of file checked.defs.hh.

12.29.1.32 #define PPL_SPECIALIZE_FUN1_0_1(name, func, ret_type, qual, type, after1)

Definition at line 191 of file checked.defs.hh.

12.29.1.33 #define PPL_SPECIALIZE_FUN1_0_2(name, func, ret_type, qual, type, after1, after2)

Definition at line 200 of file checked.defs.hh.

12.29.1.34 #define PPL_SPECIALIZE_FUN1_0_3(name, func, ret_type, qual, type, after1, after2, after3)

Definition at line 210 of file checked.defs.hh.

12.29.1.35 #define PPL_SPECIALIZE_FUN1_1_1(name, func, ret_type, before1, qual, type, after1)

Definition at line 220 of file checked.defs.hh.

12.29.1.36 #define PPL_SPECIALIZE_FUN1_1_2(name, func, ret_type, before1, qual, type, after1, after2)

Definition at line 230 of file checked.defs.hh.

12.29.1.37 #define PPL_SPECIALIZE_FUN1_2_2(name, func, ret_type, before1, before2, qual, type, after1, after2)

Definition at line 240 of file checked.defs.hh.

12.29.1.38 #define PPL_SPECIALIZE_FUN2_0_0(name, func, ret_type, qual1, type1, qual2, type2)

Definition at line 251 of file checked.defs.hh.

12.29.1.39 #define PPL_SPECIALIZE_FUN2_0_1(name, func, ret_type, qual1, type1, qual2, type2, after1)

Definition at line 261 of file checked.defs.hh.

12.29.1.40 #define PPL_SPECIALIZE_FUN2_0_2(name, func, ret_type, qual1, type1, qual2, type2, after1, after2)

Definition at line 271 of file checked.defs.hh.

12.29.1.41 #define PPL_SPECIALIZE_FUN3_0_1(name, func, ret_type, qual1, type1, qual2, type2, qual3, type3, after1)

Definition at line 282 of file checked.defs.hh.

12.29.1.42 #define PPL_SPECIALIZE_FUN5_0_1(name, func, ret_type, qual1, type1, qual2, type2, qual3, type3, qual4, type4, qual5, type5, after1)

Definition at line 295 of file checked.defs.hh.

12.29.1.43 #define PPL_SPECIALIZE_GCD(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(gcd, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 408 of file checked.defs.hh.

12.29.1.44 #define PPL_SPECIALIZE_GCDEXT(func, To1, From1, From2, To2, To3)

Value:

```
PPL_SPECIALIZE_FUN5_0_1(gcdext, func, Result, nonconst, To1,           \
                        nonconst, To2, nonconst, To3,                  \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 411 of file checked.defs.hh.

12.29.1.45 #define PPL_SPECIALIZE_IDIV(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(idiv, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 381 of file checked.defs.hh.

12.29.1.46 #define PPL_SPECIALIZE_INPUT(func, Type)

Value:

```
PPL_SPECIALIZE_FUN1_0_2(input, func, Result, nonconst, Type,           \
                        std::istream&, Rounding_Dir)
```

Definition at line 418 of file checked.defs.hh.

12.29.1.47 #define PPL_SPECIALIZE_IS_INT(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_int, func, bool, const, Type)

Definition at line 334 of file checked.defs.hh.

12.29.1.48 #define PPL_SPECIALIZE_IS_MINF(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_minf, func, bool, const, Type)

Definition at line 330 of file checked.defs.hh.

12.29.1.49 #define PPL_SPECIALIZE_IS_NAN(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_nan, func, bool, const, Type)

Definition at line 328 of file checked.defs.hh.

12.29.1.50 #define PPL_SPECIALIZE_IS_PINF(func, Type) PPL_SPECIALIZE_FUN1_0_0(is_pinf, func, bool, const, Type)

Definition at line 332 of file checked.defs.hh.

12.29.1.51 #define PPL_SPECIALIZE_LCM(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(lcm, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 415 of file checked.defs.hh.

12.29.1.52 #define PPL_SPECIALIZE_MUL(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(mul, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 372 of file checked.defs.hh.

12.29.1.53 #define PPL_SPECIALIZE_MUL_2EXP(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_2(mul_2exp, func, Result, nonconst, To,         \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 390 of file checked.defs.hh.

12.29.1.54 #define PPL_SPECIALIZE_NEG(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(neg, func, Result, nonconst, To,              \
                        const, From, Rounding_Dir)
```

Definition at line 357 of file checked.defs.hh.

12.29.1.55 #define PPL_SPECIALIZE_OUTPUT(func, Type)

Value:

```
PPL_SPECIALIZE_FUN1_1_2(output, func, Result, std::ostream&,          \
                        const, Type,                                    \
                        const Numeric_Format&, Rounding_Dir)
```

Definition at line 421 of file checked.defs.hh.

12.29.1.56 #define PPL_SPECIALIZE_REM(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(rem, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 378 of file checked.defs.hh.

12.29.1.57 #define PPL_SPECIALIZE_SGN(func, From) PPL_SPECIALIZE_FUN1_0_0(sgn, func, Result_Relation, const, From)

Definition at line 322 of file checked.defs.hh.

12.29.1.58 #define PPL_SPECIALIZE_SMOD_2EXP(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_2(smod_2exp, func, Result, nonconst, To,  \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 396 of file checked.defs.hh.

12.29.1.59 #define PPL_SPECIALIZE_SQRT(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(sqrt, func, Result, nonconst, To,             \
                        const, From, Rounding_Dir)
```

Definition at line 363 of file checked.defs.hh.

12.29.1.60 #define PPL_SPECIALIZE_SUB(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(sub, func, Result, nonconst, To,              \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 369 of file checked.defs.hh.

12.29.1.61 #define PPL_SPECIALIZE_SUB_2EXP(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_2(sub_2exp, func, Result, nonconst, To,         \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 387 of file checked.defs.hh.

12.29.1.62 #define PPL_SPECIALIZE_SUB_MUL(func, To, From1, From2)

Value:

```
PPL_SPECIALIZE_FUN3_0_1(sub_mul, func, Result, nonconst, To,          \
                        const, From1, const, From2, Rounding_Dir)
```

Definition at line 405 of file checked.defs.hh.

12.29.1.63 #define PPL_SPECIALIZE_TRUNC(func, To, From)

Value:

```
PPL_SPECIALIZE_FUN2_0_1(trunc, func, Result, nonconst, To,            \
                        const, From, Rounding_Dir)
```

Definition at line 354 of file checked.defs.hh.

12.29.1.64 #define PPL_SPECIALIZE_UMOD_2EXP(func, To, From)

Value:

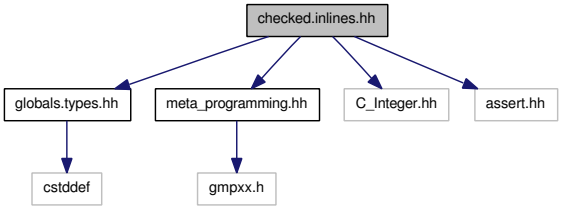
```
PPL_SPECIALIZE_FUN2_0_2(umod_2exp, func, Result, nonconst, To,  \
                        const, From, unsigned int, Rounding_Dir)
```

Definition at line 399 of file checked.defs.hh.

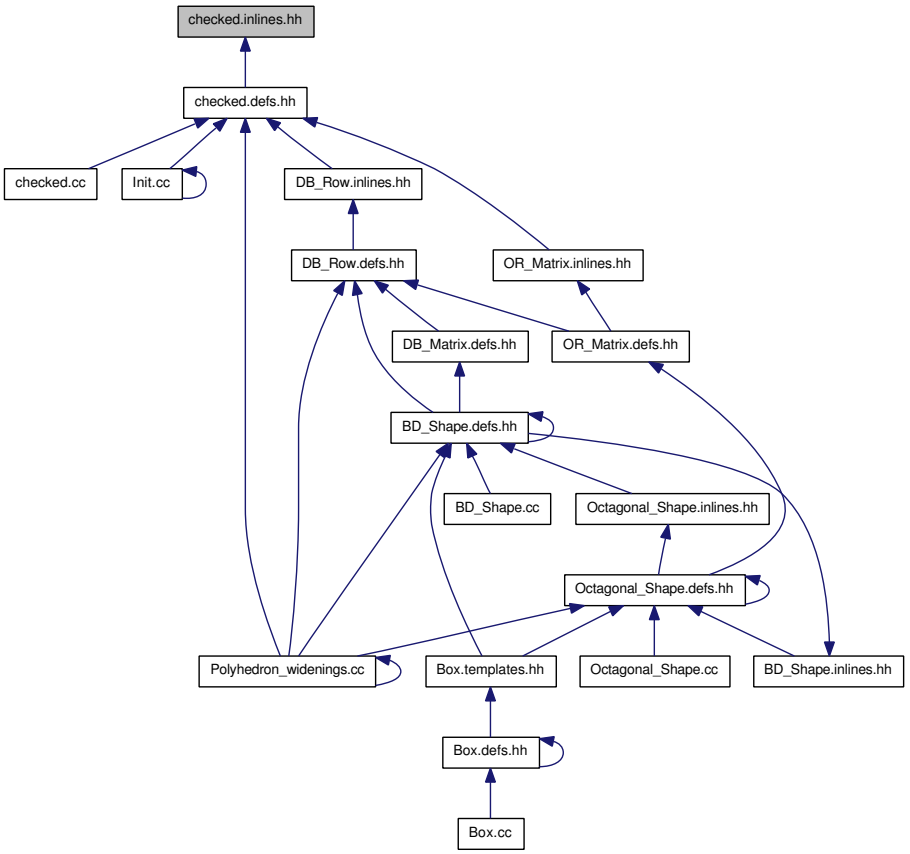
12.30 checked.inlines.hh File Reference

```
#include "globals.types.hh"
#include "meta_programming.hh"
#include "C_Integer.hh"
#include "assert.hh"
```

Include dependency graph for checked.inlines.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Checked::Safe_Conversion< T1, T2 >
- struct Parma_Polyhedra_Library::Checked::Safe_Conversion< T, T >
- struct Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, Enable >
- struct Parma_Polyhedra_Library::Checked::Safe_Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Defines

- #define PPL_LT_SILENT(a, b) ((a) < (b))

Performs the test `a < b` *avoiding the warning about the comparison being always false due to limited range of data type. FIXME: we don't have found a working solution. gcc option -Wno-type-limits suppress the warning.*

- #define PPL_GT_SILENT(a, b) ((a) > (b))
- #define PPL_SAFE_CONVERSION(To, From) template <> struct Safe_Conversion<To, From> : public True { }
- #define PPL_MATCH_GMP_GCDEXT 1

Functions

- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed short, signed char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed long long, signed char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (signed long long, signed short)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned short, char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned short, unsigned char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long, char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long, unsigned char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long long, char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (unsigned long long, unsigned char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (double, float)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (long double, float)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (long double, double)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, signed char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpz_class, unsigned char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, signed char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, unsigned char)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, float)
- Parma_Polyhedra_Library::Checked::PPL_SAFE_CONVERSION (mpq_class, double)
- template<typename Policy , typename Type >
  struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct)< Policy
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct)
- template<typename To_Policy , typename To >
  struct Parma_Polyhedra_Library::Checked::PPL_FUNCTION_CLASS (construct_special)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_exact (To &to, const From &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Enable_If< Is_Same< To_Policy, From_Policy >::value, void >::type Parma_Polyhedra_Library::Checked::copy_generic (Type &to, const Type &from)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::abs_generic (To &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From >
  void Parma_Polyhedra_Library::Checked::gcd_exact_noabs (To &to, const From &x, const From &y)

- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 >
  Result Parma_Polyhedra_Library::Checked::gcd_exact (To &to, const From1 &x, const From2 &y, Rounding_Dir dir)
- template<typename To1_Policy , typename To2_Policy , typename To3_Policy , typename From1_Policy , typename From2_Policy , typename To1 , typename To2 , typename To3 , typename From1 , typename From2 >
  Result Parma_Polyhedra_Library::Checked::gcdext_exact (To1 &to, To2 &s, To3 &t, const From1 &x, const From2 &y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename To , typename From1 , typename From2 >
  Result Parma_Polyhedra_Library::Checked::lcm_gcd_exact (To &to, const From1 &x, const From2 &y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result_Relation Parma_Polyhedra_Library::Checked::sgn_generic (const Type &x)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type Parma_Polyhedra_Library::Checked::lt (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type Parma_Polyhedra_Library::Checked::le (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<(Safe_Int_Comparison< T1, T2 >::value||Safe_Conversion< T1, T2 >::value||Safe_Conversion< T2, T1 >::value), bool >::type Parma_Polyhedra_Library::Checked::eq (const T1 &x, const T2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool Parma_Polyhedra_Library::Checked::lt_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool Parma_Polyhedra_Library::Checked::le_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  bool Parma_Polyhedra_Library::Checked::eq_p (const Type1 &x, const Type2 &y)
- template<typename Policy1 , typename Policy2 , typename Type1 , typename Type2 >
  Result_Relation Parma_Polyhedra_Library::Checked::cmp_generic (const Type1 &x, const Type2 &y)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::assign_nan (Type &to, Result r)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::input_generic (Type &to, std::istream &is, Rounding_Dir dir)

Variables

- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type Parma_Polyhedra_Library::Checked::lt (const S &x, const U &y)
- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type Parma_Polyhedra_Library::Checked::le (const S &x, const U &y)
- Enable_If<(!Safe_Int_Comparison< S, U >::value &&C_Integer< U >::value &&C_Integer< S >::is_signed), bool >::type Parma_Polyhedra_Library::Checked::eq (const S &x, const U &y)

12.30.1 Define Documentation

12.30.1.1 #define PPL_GT_SILENT(a, b) ((a) > (b))

Definition at line 39 of file checked.inlines.hh.

Referenced by Parma_Polyhedra_Library::Checked::assign_signed_int_mpz(), Parma_Polyhedra_Library::Checked::assign_signed_int_signed_int(), Parma_Polyhedra_Library::Checked::assign_unsigned_int_mpz(), Parma_Polyhedra_Library::Checked::assign_unsigned_int_unsigned_int(), Parma_Polyhedra_Library::Checked::mul_2exp_signed_int(), and Parma_Polyhedra_Library::Checked::mul_2exp_unsigned_int().

12.30.1.2 #define PPL_LT_SILENT(a, b) ((a) < (b))

Performs the test `a < b` avoiding the warning about the comparison being always false due to limited range of data type. FIXME: we don't have found a working solution. gcc option -Wno-type-limits suppress the warning.

Definition at line 38 of file checked.inlines.hh.

Referenced by Parma_Polyhedra_Library::Checked::assign_signed_int_mpz(), Parma_Polyhedra_Library::Checked::assign_signed_int_signed_int(), and Parma_Polyhedra_Library::Checked::mul_2exp_signed_int().

12.30.1.3 #define PPL_MATCH_GMP_GCDEXT 1

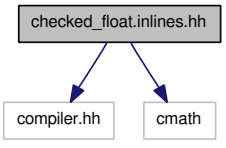
12.30.1.4 #define PPL_SAFE_CONVERSION(To, From) template <> struct Safe_Conversion<To, From> : public True { }

Definition at line 52 of file checked.inlines.hh.

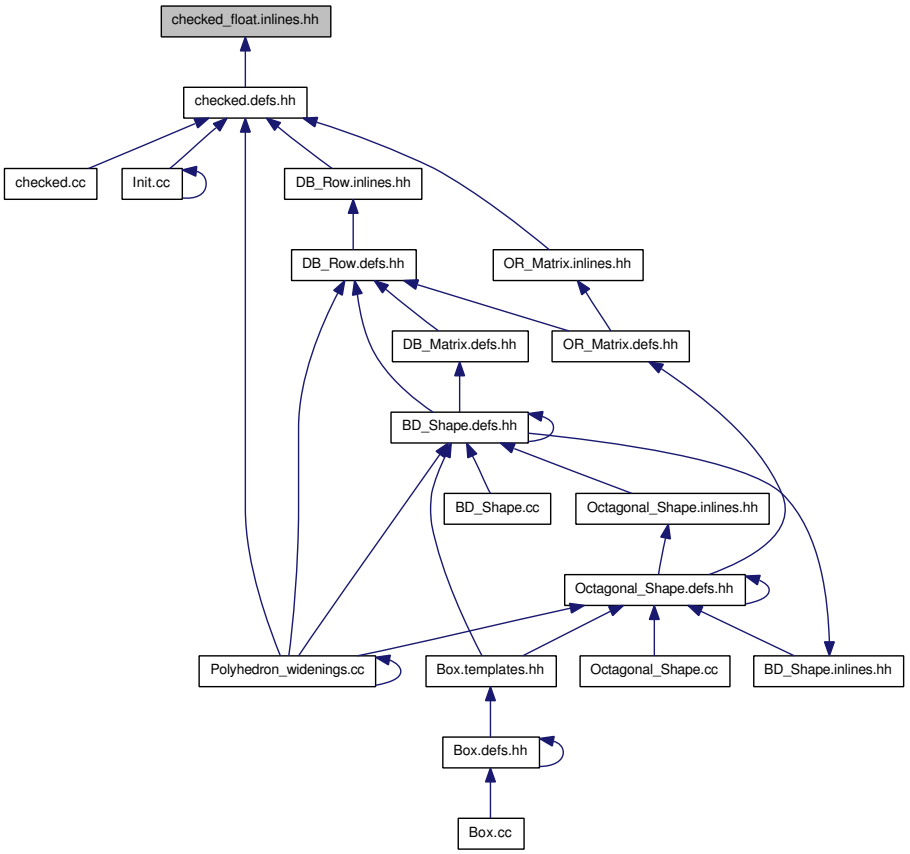
12.31 checked_float.inlines.hh File Reference

```
#include "compiler.hh"
#include <cmath>
```

Include dependency graph for checked_float.inlines.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Checked::Float_2exp

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Functions

- float Parma_Polyhedra_Library::Checked::fma (float x, float y, float z)
- double Parma_Polyhedra_Library::Checked::fma (double x, double y, double z)
- long double Parma_Polyhedra_Library::Checked::fma (long double x, long double y, long double z)
- double Parma_Polyhedra_Library::Checked::rint (double x)
- long double Parma_Polyhedra_Library::Checked::rint (long double x)

- bool Parma_Polyhedra_Library::Checked::fpu_direct_rounding (Rounding_Dir dir)
- bool Parma_Polyhedra_Library::Checked::fpu_inverse_rounding (Rounding_Dir dir)
- void Parma_Polyhedra_Library::Checked::limit_precision (const float &v)
- void Parma_Polyhedra_Library::Checked::limit_precision (const double &v)
- void Parma_Polyhedra_Library::Checked::limit_precision (const long double &)
- template<typename Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::classify_float (const T v, bool nan, bool inf, bool sign)
- template<typename Policy , typename T >
  bool Parma_Polyhedra_Library::Checked::is_nan_float (const T v)
- template<typename Policy , typename T >
  int Parma_Polyhedra_Library::Checked::is_inf_float (const T v)
- template<typename Policy , typename T >
  bool Parma_Polyhedra_Library::Checked::is_minf_float (const T v)
- template<typename Policy , typename T >
  bool Parma_Polyhedra_Library::Checked::is_pinf_float (const T v)
- template<typename Policy , typename T >
  bool Parma_Polyhedra_Library::Checked::is_int_float (const T v)
- template<typename Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::assign_special_float (T &v, Result_Class c, Rounding_Dir)
- template<typename T >
  void Parma_Polyhedra_Library::Checked::pred_float (T &v)
- template<typename T >
  void Parma_Polyhedra_Library::Checked::succ_float (T &v)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_lt_float (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_gt_float (To &to, Rounding_Dir dir)
- template<typename Policy >
  void Parma_Polyhedra_Library::Checked::prepare_inexact (Rounding_Dir dir)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::result_relation (Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_float_float_exact (To &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_float_float_inexact (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_float_float (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::floor_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::ceil_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::trunc_float (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::neg_float (Type &to, const Type from, Rounding_Dir)

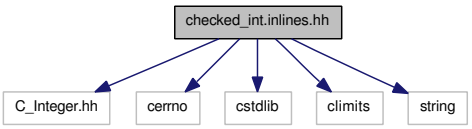
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::idiv_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::rem_float (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::smod_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::umod_2exp_float (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::abs_float (Type &to, const Type from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sqrt_float (Type &to, const Type from, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result_Relation Parma_Polyhedra_Library::Checked::sgn_float (const Type x)
- template<typename Policy1 , typename Policy2 , typename Type >
  Result_Relation Parma_Polyhedra_Library::Checked::cmp_float (const Type x, const Type y)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_float_int_inexact (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_float_int (To &to, const From from, Rounding_-Dir dir)
- template<typename Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::set_neg_overflow_float (T &to, Rounding_Dir dir)

- template<typename Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::set_pos_overflow_float (T &to, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::assign_float_mpz (T &to, const mpz_class &_from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename T >
  Result Parma_Polyhedra_Library::Checked::assign_float_mpq (T &to, const mpq_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_mul_float (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::output_float (std::ostream &os, const Type from, const Numeric_Format &, Rounding_Dir)

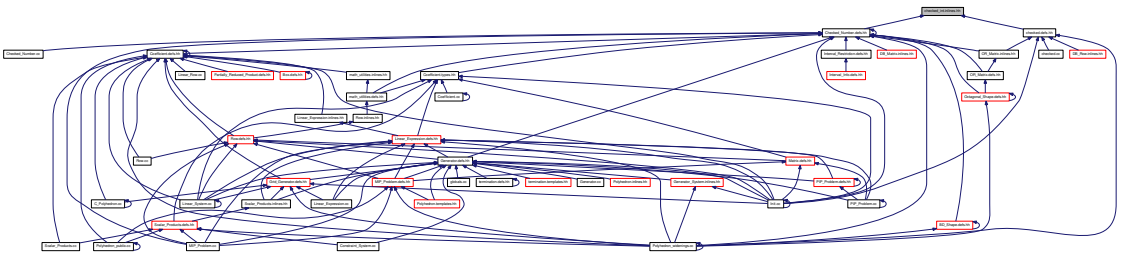
12.32 checked_int.inlines.hh File Reference

```
#include "C_Integer.hh"
#include <cerrno>
#include <cstdlib>
#include <climits>
#include <string>
```

Include dependency graph for checked_int.inlines.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Checked::Extended_Int< Policy, Type >
- struct Parma_Polyhedra_Library::Checked::Larger< char >

- struct Parma_Polyhedra_Library::Checked::Larger< signed char >
- struct Parma_Polyhedra_Library::Checked::Larger< unsigned char >
- struct Parma_Polyhedra_Library::Checked::Larger< signed short >
- struct Parma_Polyhedra_Library::Checked::Larger< unsigned short >
- struct Parma_Polyhedra_Library::Checked::Larger< signed int >
- struct Parma_Polyhedra_Library::Checked::Larger< unsigned int >
- struct Parma_Polyhedra_Library::Checked::Larger< signed long >
- struct Parma_Polyhedra_Library::Checked::Larger< unsigned long >
- struct Parma_Polyhedra_Library::Checked::Larger< signed long long >
- struct Parma_Polyhedra_Library::Checked::Larger< unsigned long long >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Defines

- #define PPL_ASSIGN2_SIGNED_SIGNED(Smaller, Larger)
- #define PPL_ASSIGN2_UNSIGNED_UNSIGNED(Smaller, Larger)
- #define PPL_ASSIGN2_UNSIGNED_SIGNED(Smaller, Larger)
- #define PPL_ASSIGN2_SIGNED_UNSIGNED(Smaller, Larger)
- #define PPL_ASSIGN_SIGNED(Type) PPL_SPECIALIZE_ASSIGN(assign_signed_int_signed_int, Type, Type)
- #define PPL_ASSIGN_UNSIGNED(Type) PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_unsigned_int, Type, Type)

Typedefs

- typedef int16_t Parma_Polyhedra_Library::Checked::int_fast16_t
- typedef int32_t Parma_Polyhedra_Library::Checked::int_fast32_t
- typedef int64_t Parma_Polyhedra_Library::Checked::int_fast64_t
- typedef uint16_t Parma_Polyhedra_Library::Checked::uint_fast16_t
- typedef uint32_t Parma_Polyhedra_Library::Checked::uint_fast32_t
- typedef uint64_t Parma_Polyhedra_Library::Checked::uint_fast64_t

Functions

- signed long long strtoll (const char *nptr, char **endptr, int base)
- unsigned long long strtoull (const char *nptr, char **endptr, int base)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::set_neg_overflow_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::set_pos_overflow_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_lt_int_no_overflow (To &to, Rounding_Dir dir)

- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_gt_int_no_overflow (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_lt_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::round_gt_int (To &to, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::classify_int (const Type v, bool nan, bool inf, bool sign)
- template<typename Policy , typename Type >
  bool Parma_Polyhedra_Library::Checked::is_nan_int (const Type v)
- template<typename Policy , typename Type >
  bool Parma_Polyhedra_Library::Checked::is_minf_int (const Type v)
- template<typename Policy , typename Type >
  bool Parma_Polyhedra_Library::Checked::is_pinf_int (const Type v)
- template<typename Policy , typename Type >
  bool Parma_Polyhedra_Library::Checked::is_int_int (const Type v)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::assign_special_int (Type &v, Result_Class c, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_signed_int_signed_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_signed_int_unsigned_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_signed_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_unsigned_int (To &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_int_float (To &to, const From from, Rounding_-Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::assign_signed_int_mpz (To &to, const mpz_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::assign_unsigned_int_mpz (To &to, const mpz_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename To >
  Result Parma_Polyhedra_Library::Checked::assign_int_mpq (To &to, const mpq_class &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::neg_int_larger (Type &to, const Type x, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)

- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_int_larger (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::neg_signed_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::neg_unsigned_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::idiv_signed_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::idiv_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::rem_signed_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::rem_unsigned_int (Type &to, const Type x, const Type y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::div_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)

- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::mul_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::smod_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::smod_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::umod_2exp_unsigned_int (Type &to, const Type x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::umod_2exp_signed_int (Type &to, const Type x, unsigned int exp, Rounding_Dir dir)
- template<typename Type >
  void Parma_Polyhedra_Library::Checked::isqrtrem (Type &q, Type &r, const Type from)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sqrt_unsigned_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sqrt_signed_int (Type &to, const Type from, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::add_mul_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::sub_mul_int (Type &to, const Type x, const Type y, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::output_char (std::ostream &os, Type &from, const Numeric_Format &, Rounding_Dir)
- template<typename Policy , typename Type >
  Result Parma_Polyhedra_Library::Checked::output_int (std::ostream &os, Type &from, const Numeric_Format &, Rounding_Dir)

12.32.1 Define Documentation

12.32.1.1 #define PPL_ASSIGN2_SIGNED_SIGNED(Smaller, Larger)

Value:

```
PPL_SPECIALIZE_ASSIGN(assign_signed_int_signed_int, Smaller, Larger) \
PPL_SPECIALIZE_ASSIGN(assign_signed_int_signed_int, Larger, Smaller)
```

Definition at line 409 of file checked_int.inlines.hh.

12.32.1.2 #define PPL_ASSIGN2_SIGNED_UNSIGNED(Smaller, Larger)

Value:

```
PPL_SPECIALIZE_ASSIGN(assign_signed_int_unsigned_int, Smaller, Larger) \
PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_signed_int, Larger, Smaller)
```

Definition at line 421 of file checked_int.inlines.hh.

12.32.1.3 #define PPL_ASSIGN2_UNSIGNED_SIGNED(Smaller, Larger)

Value:

```
PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_signed_int, Smaller, Larger) \
PPL_SPECIALIZE_ASSIGN(assign_signed_int_unsigned_int, Larger, Smaller)
```

Definition at line 417 of file checked_int.inlines.hh.

12.32.1.4 #define PPL_ASSIGN2_UNSIGNED_UNSIGNED(Smaller, Larger)

Value:

```
PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_unsigned_int, Smaller, Larger) \
PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_unsigned_int, Larger, Smaller)
```

Definition at line 413 of file checked_int.inlines.hh.

12.32.1.5 #define PPL_ASSIGN_SIGNED(Type) PPL_SPECIALIZE_ASSIGN(assign_signed_int_signed_int, Type, Type)

Definition at line 425 of file checked_int.inlines.hh.

12.32.1.6 #define PPL_ASSIGN_UNSIGNED(Type) PPL_SPECIALIZE_ASSIGN(assign_unsigned_int_unsigned_int, Type, Type)

Definition at line 427 of file checked_int.inlines.hh.

12.32.2 Function Documentation

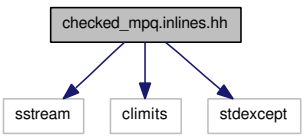
12.32.2.1 signed long long strtoll (const char * *nptr*, char ** *endptr*, int *base*)

12.32.2.2 unsigned long long strtoull (const char * *nptr*, char ** *endptr*, int *base*)

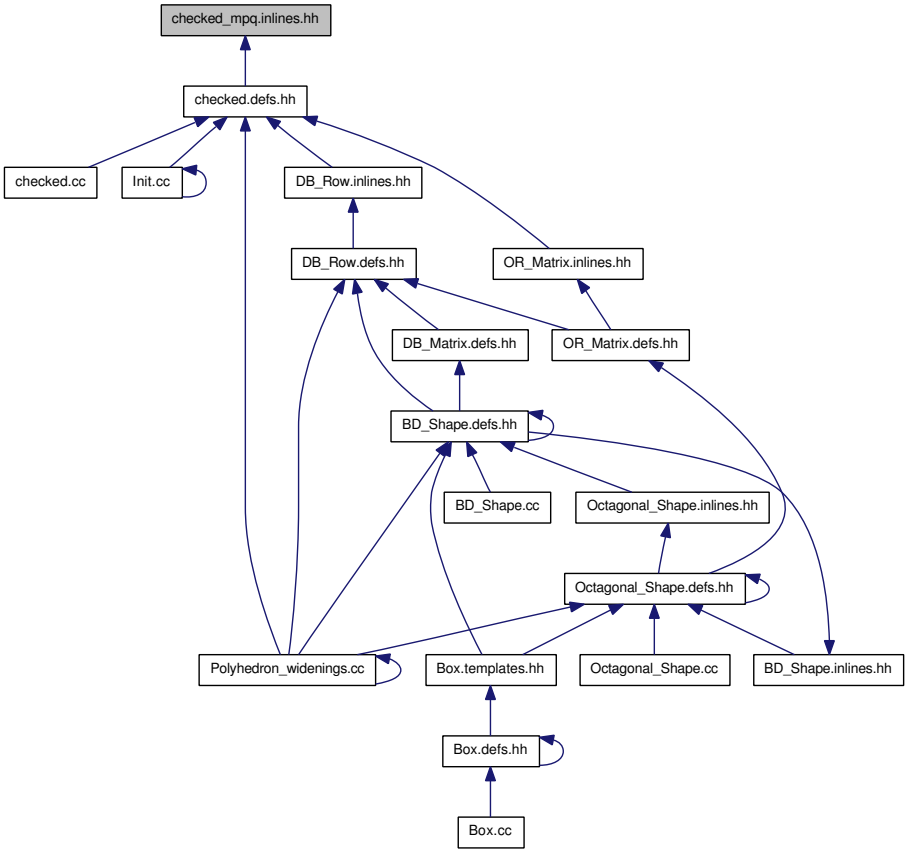
12.33 checked_mpq.inlines.hh File Reference

```
#include <sstream>
#include <climits>
#include <stdexcept>
```

Include dependency graph for checked_mpq.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Functions

- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::classify_mpq (const mpq_class &v, bool nan, bool inf, bool sign)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_nan_mpq (const mpq_class &v)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_minf_mpq (const mpq_class &v)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_pinf_mpq (const mpq_class &v)

- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_int_mpq (const mpq_class &v)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::assign_special_mpq (mpq_class &v, Result_Class c, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::construct_mpq_base (mpq_class &to, const From &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::construct_mpq_float (mpq_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpq_float (mpq_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpq_signed_int (mpq_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpq_unsigned_int (mpq_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::floor_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::ceil_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::trunc_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::neg_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::add_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::div_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::idiv_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::rem_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::add_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)

- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::mul_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::div_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::smod_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::umod_2exp_mpq (mpq_class &to, const mpq_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::abs_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::add_mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_mul_mpq (mpq_class &to, const mpq_class &x, const mpq_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::sqrt_mpq (mpq_class &to, const mpq_class &from, Rounding_Dir dir)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::input_mpq (mpq_class &to, std::istream &is, Rounding_Dir dir)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::output_mpq (std::ostream &os, const mpq_class &from, const Numeric_Format &, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpq_long_double (mpq_class &to, const From &from, Rounding_Dir dir)
- unsigned Parma_Polyhedra_Library::irrational_precision ()

  Returns the precision parameter used for irrational calculations.

- void Parma_Polyhedra_Library::set_irrational_precision (const unsigned p)

  Sets the precision parameter used for irrational calculations.

Variables

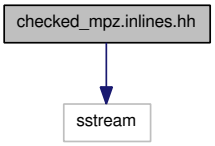
- unsigned Parma_Polyhedra_Library::Checked::irrational_precision

  Holds the precision parameter used for irrational calculations.

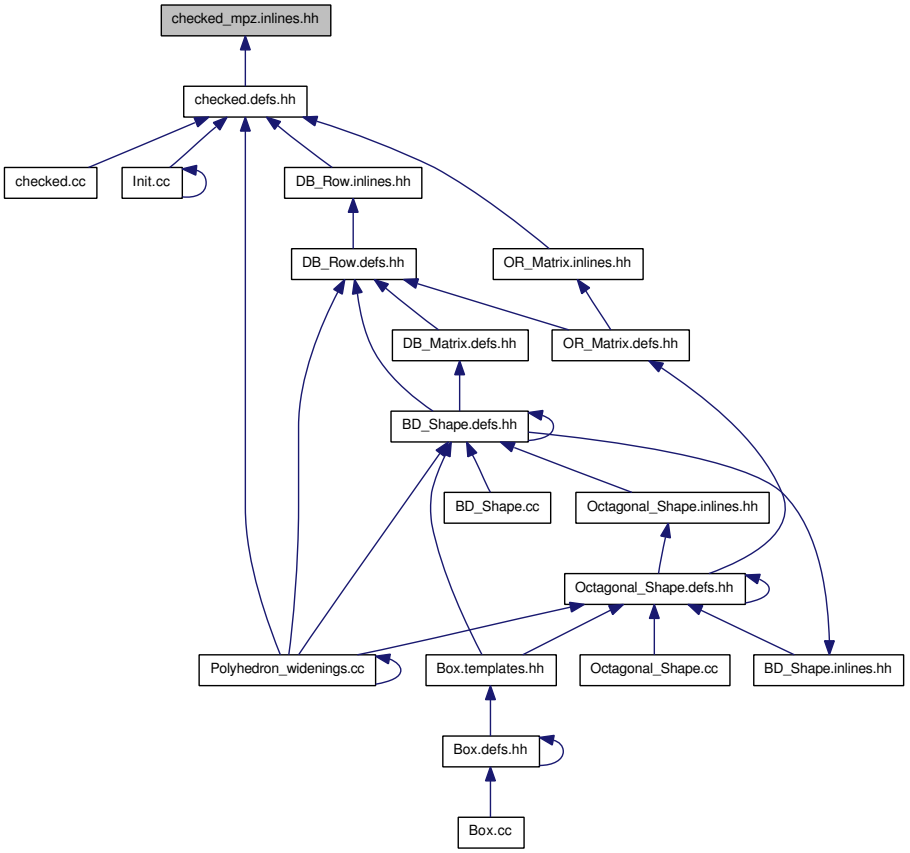
12.34 checked_mpz.inlines.hh File Reference

```
#include <sstream>
```

Include dependency graph for checked_mpz.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

Typedefs

- typedef int Parma_Polyhedra_Library::Checked::mp_size_field_t

  This is assumed to be the type of the _mp_size field of GMP's __mpz_struct.

Functions

- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::round_lt_mpz (mpz_class &to, Rounding_Dir dir)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::round_gt_mpz (mpz_class &to, Rounding_Dir dir)
- mp_size_field_t Parma_Polyhedra_Library::Checked::get_mp_size (const mpz_class &v)
- void Parma_Polyhedra_Library::Checked::set_mp_size (mpz_class &v, mp_size_field_t size)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::classify_mpz (const mpz_class &v, bool nan, bool inf, bool sign)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_nan_mpz (const mpz_class &v)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_minf_mpz (const mpz_class &v)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_pinf_mpz (const mpz_class &v)
- template<typename Policy >
  bool Parma_Polyhedra_Library::Checked::is_int_mpz (const mpz_class &v)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::assign_special_mpz (mpz_class &v, Result_Class c, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  void Parma_Polyhedra_Library::Checked::copy_mpz (mpz_class &to, const mpz_class &from)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::construct_mpz_base (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::construct_mpz_float (mpz_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpz_signed_int (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpz_unsigned_int (mpz_class &to, const From from, Rounding_Dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpz_float (mpz_class &to, const From from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy , typename From >
  Result Parma_Polyhedra_Library::Checked::assign_mpz_long_double (mpz_class &to, const From &from, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::assign_mpz_mpq (mpz_class &to, const mpq_class &from, Rounding_Dir dir)

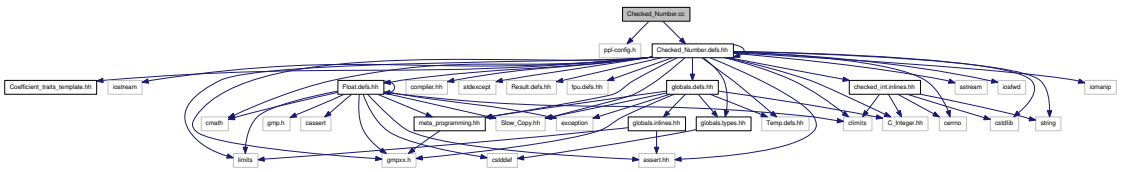
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::neg_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::add_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::div_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::idiv_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::rem_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::add_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::mul_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::div_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::smod_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::umod_2exp_mpz (mpz_class &to, const mpz_class &x, unsigned int exp, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::abs_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::add_mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::sub_mul_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::gcd_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)

- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::gcdext_mpz (mpz_class &to, mpz_class &s, mpz_class &t, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From1_Policy , typename From2_Policy >
  Result Parma_Polyhedra_Library::Checked::lcm_mpz (mpz_class &to, const mpz_class &x, const mpz_class &y, Rounding_Dir)
- template<typename To_Policy , typename From_Policy >
  Result Parma_Polyhedra_Library::Checked::sqrt_mpz (mpz_class &to, const mpz_class &from, Rounding_Dir dir)
- template<typename Policy , typename Type >
  Result_Relation Parma_Polyhedra_Library::Checked::sgn_mp (const Type &x)
- template<typename Policy1 , typename Policy2 , typename Type >
  Result_Relation Parma_Polyhedra_Library::Checked::cmp_mp (const Type &x, const Type &y)
- template<typename Policy >
  Result Parma_Polyhedra_Library::Checked::output_mpz (std::ostream &os, const mpz_class &from, const Numeric_Format &, Rounding_Dir)

12.35 Checked_Number.cc File Reference

```
#include <ppl-config.h>
#include "Checked_Number.defs.hh"
```

Include dependency graph for Checked_Number.cc:



Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

Functions

- void Parma_Polyhedra_Library::throw_result_exception (Result r)

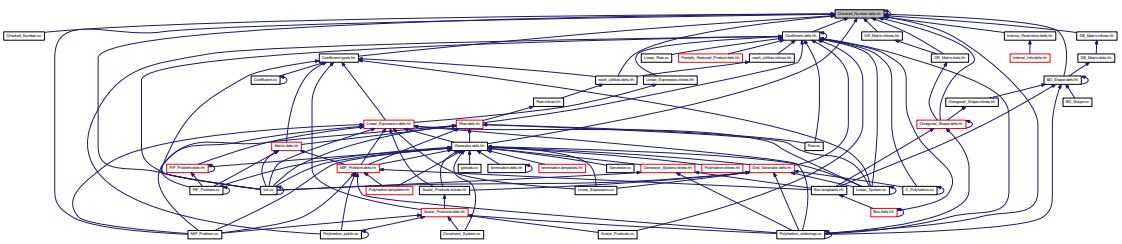
12.36 Checked_Number.defs.hh File Reference

```
#include "Checked_Number.types.hh"
#include "Coefficient_traits_template.hh"
#include <iostream>
#include <gmpxx.h>
#include <limits>
```

```
#include "Temp.defs.hh"
#include "assert.hh"
#include "compiler.hh"
#include <stdexcept>
#include "Result.defs.hh"
#include "fpu.defs.hh"
#include "Float.defs.hh"
#include "globals.types.hh"
#include "meta_programming.hh"
#include "C_Integer.hh"
#include <cerrno>
#include <cstdlib>
#include <climits>
#include <string>
#include <cmath>
#include <sstream>
#include "Slow_Copy.hh"
#include <iosfwd>
#include "globals.defs.hh"
#include "checked_int.inlines.hh"
#include <iomanip>
```

Include dependency graph for Checked_Number.defs.hh:

This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Extended_Number_Policy

- struct Parma_Polyhedra_Library::Check_Overflow_Policy< T >

  A policy checking for overflows.

- struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >
- struct Parma_Polyhedra_Library::Native_Checked_To_Wrapper< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Is_Checked< T >
- struct Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::Is_Native_Or_Checked< T >
- class Parma_Polyhedra_Library::Checked_Number< T, Policy >

  A wrapper for numeric types implementing a given policy.

- struct Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >
- struct Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Defines

- #define PPL_DECLARE_FUNC1_A(name)
- #define PPL_DECLARE_FUNC1_B(name)
- #define PPL_DECLARE_FUNC2(name)
- #define PPL_DECLARE_FUNC4(name)

Functions

- void Parma_Polyhedra_Library::throw_result_exception (Result r)
- template<typename T >
  T Parma_Polyhedra_Library::plus_infinity ()
- template<typename T >
  T Parma_Polyhedra_Library::minus_infinity ()
- template<typename T >
  T Parma_Polyhedra_Library::not_a_number ()
- template<typename T >
  void Parma_Polyhedra_Library::maybe_reset_fpu_inexact ()
- template<typename T >
  int Parma_Polyhedra_Library::maybe_check_fpu_inexact ()

Relational Operators and Comparison Functions

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::equal (const T1 &x, const T2 &y)

- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::not_equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::greater_or_equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::greater_than (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::less_or_equal (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Native_Or_Checked< T1 >::value &&Is_Native_Or_Checked< T2 >::value, bool >::type Parma_Polyhedra_Library::less_than (const T1 &x, const T2 &y)

Input-Output Operators

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::ascii_dump (std::ostream &s, const T &t)
  Ascii dump for native or checked.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::ascii_load (std::ostream &s, T &t)
  Ascii load for native or checked.

12.36.1 Define Documentation

12.36.1.1 #define PPL_DECLARE_FUNC1_A(name)

Value:

```
template <typename To, typename From> \
typename Enable_If<Is_Native_Or_Checked<To>::value \
                   && Is_Native_Or_Checked<From>::value, \
                   Result>::type \
name(To& to, const From& x, Rounding_Dir dir);
```

Definition at line 478 of file Checked_Number.defs.hh.

12.36.1.2 #define PPL_DECLARE_FUNC1_B(name)

Value:

```
template <typename To, typename From> \
typename Enable_If<Is_Native_Or_Checked<To>::value \
                   && Is_Native_Or_Checked<From>::value, \
                   Result>::type \
name(To& to, const From& x, int exp, Rounding_Dir dir);
```

Definition at line 495 of file Checked_Number.defs.hh.

12.36.1.3 #define PPL_DECLARE_FUNC2(name)

Definition at line 511 of file Checked_Number.defs.hh.

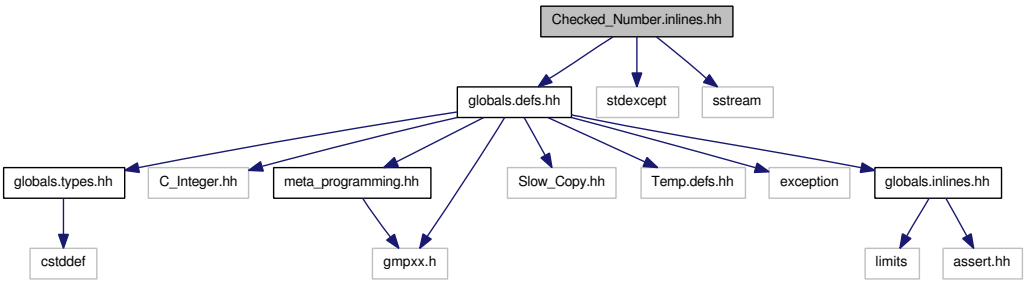
12.36.1.4 #define PPL_DECLARE_FUNC4(name)

Definition at line 532 of file Checked_Number.defs.hh.

12.37 Checked_Number.inlines.hh File Reference

```
#include "globals.defs.hh"
#include <stdexcept>
#include <sstream>
```

Include dependency graph for Checked_Number.inlines.hh:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Defines

- #define DEBUG_ROUND_NOT_NEEDED
- #define PPL_DEFINE_CTOR(type)
- #define PPL_COND_0(...)
- #define PPL_COND_1(...) __VA_ARGS__
- #define PPL_COND_(if,...) PPL_COND_##if(__VA_ARGS__)
- #define PPL_COND(if,...) PPL_COND_(if, __VA_ARGS__)
- #define PPL_DEFINE_FUNC1_A(name, func)
- #define PPL_DEFINE_FUNC1_B(name, func)
- #define PPL_DEFINE_FUNC2(name, func)
- #define PPL_DEFINE_FUNC4(name, func)
- #define PPL_DEFINE_INCREMENT(f, fun)
- #define PPL_DEFINE_BINARY_OP_ASSIGN(f, fun)

- #define PPL_DEFINE_BINARY_OP(f, fun)
- #define PPL_DEFINE_COMPARE_OP(f, fun)
- #define PPL_DEFINE_COMPARE(f, fun)
- #define PPL_DEFINE_ASSIGN_FUN2_1(f, fun)
- #define PPL_DEFINE_ASSIGN_FUN2_2(f, fun)
- #define PPL_DEFINE_ASSIGN_FUN3_3(f, fun)
- #define PPL_DEFINE_ASSIGN_FUN5_5(f, fun)
- #define PPL_DEFINE_ASSIGN_2EXP(f, fun)

Functions

- Rounding_Dir Parma_Polyhedra_Library::rounding_dir (Rounding_Dir dir)
- Result Parma_Polyhedra_Library::check_result (Result r, Rounding_Dir dir)
- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type Parma_Polyhedra_Library::assign_r (To &to, const From &, Rounding_Dir dir)
- template<typename To , typename From >
  Enable_If< Is_Native_Or_Checked< To >::value &&Is_Special< From >::value, Result >::type Parma_Polyhedra_Library::construct (To &to, const From &, Rounding_Dir dir)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_minus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_plus_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, int >::type Parma_Polyhedra_Library::is_infinity (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_not_a_number (const T &x)
- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_integer (const T &x)
- template<typename T , typename Policy >
  bool Parma_Polyhedra_Library::is_not_a_number (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  bool Parma_Polyhedra_Library::is_minus_infinity (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  bool Parma_Polyhedra_Library::is_plus_infinity (const Checked_Number< T, Policy > &x)
- template<typename T , typename Policy >
  void Parma_Polyhedra_Library::exact_div_assign (Checked_Number< T, Policy > &x, const Checked_Number< T, Policy > &y, const Checked_Number< T, Policy > &z)
- template<typename T >
  T Parma_Polyhedra_Library::plus_infinity ()
- template<typename T >
  T Parma_Polyhedra_Library::minus_infinity ()
- template<typename T >
  T Parma_Polyhedra_Library::not_a_number ()
- template<typename T >
  void Parma_Polyhedra_Library::maybe_reset_fpu_inexact ()
- template<typename T >
  int Parma_Polyhedra_Library::maybe_check_fpu_inexact ()

12.37.1 Define Documentation

12.37.1.1 #define DEBUG_ROUND_NOT_NEEDED

Definition at line 34 of file Checked_Number.inlines.hh.

12.37.1.2 #define PPL_COND(if, ...) PPL_COND_(if, __VA_ARGS__)

Definition at line 135 of file Checked_Number.inlines.hh.

12.37.1.3 #define PPL_COND_(if, ...) PPL_COND_##if(__VA_ARGS__)

Definition at line 134 of file Checked_Number.inlines.hh.

12.37.1.4 #define PPL_COND_0(...)

Definition at line 132 of file Checked_Number.inlines.hh.

12.37.1.5 #define PPL_COND_1(...) __VA_ARGS__

Definition at line 133 of file Checked_Number.inlines.hh.

12.37.1.6 #define PPL_DEFINE_ASSIGN_2EXP(f, fun)

Definition at line 697 of file Checked_Number.inlines.hh.

12.37.1.7 #define PPL_DEFINE_ASSIGN_FUN2_1(f, fun)

Value:

```
template <typename T, typename Policy> \
inline void \
f(Checked_Number<T, Policy>& x) { \
  Policy::handle_result(fun(x, x, Policy::ROUND_DEFAULT_FUNCTION));     \
}
```

Definition at line 630 of file Checked_Number.inlines.hh.

12.37.1.8 #define PPL_DEFINE_ASSIGN_FUN2_2(f, fun)

Value:

```
template <typename T, typename Policy> \
inline void \
f(Checked_Number<T, Policy>& x, const Checked_Number<T, Policy>& y) { \
  Policy::handle_result(fun(x, y, Policy::ROUND_DEFAULT_FUNCTION)); \
}
```

Definition at line 637 of file Checked_Number.inlines.hh.

12.37.1.9 #define PPL_DEFINE_ASSIGN_FUN3_3(f, fun)

Definition at line 644 of file Checked_Number.inlines.hh.

12.37.1.10 #define PPL_DEFINE_ASSIGN_FUN5_5(f, fun)

Definition at line 652 of file Checked_Number.inlines.hh.

12.37.1.11 #define PPL_DEFINE_BINARY_OP(f, fun)

Definition at line 537 of file Checked_Number.inlines.hh.

12.37.1.12 #define PPL_DEFINE_BINARY_OP_ASSIGN(f, fun)

Definition at line 502 of file Checked_Number.inlines.hh.

12.37.1.13 #define PPL_DEFINE_COMPARE(f, fun)

Definition at line 593 of file Checked_Number.inlines.hh.

12.37.1.14 #define PPL_DEFINE_COMPARE_OP(f, fun)

Definition at line 570 of file Checked_Number.inlines.hh.

12.37.1.15 #define PPL_DEFINE_CTOR(type)

Definition at line 113 of file Checked_Number.inlines.hh.

12.37.1.16 #define PPL_DEFINE_FUNC1_A(name, func)

Definition at line 347 of file Checked_Number.inlines.hh.

12.37.1.17 #define PPL_DEFINE_FUNC1_B(name, func)

Definition at line 373 of file Checked_Number.inlines.hh.

12.37.1.18 #define PPL_DEFINE_FUNC2(name, func)

Definition at line 399 of file Checked_Number.inlines.hh.

12.37.1.19 #define PPL_DEFINE_FUNC4(name, func)

Definition at line 432 of file Checked_Number.inlines.hh.

12.37.1.20 #define PPL_DEFINE_INCREMENT(f, fun)

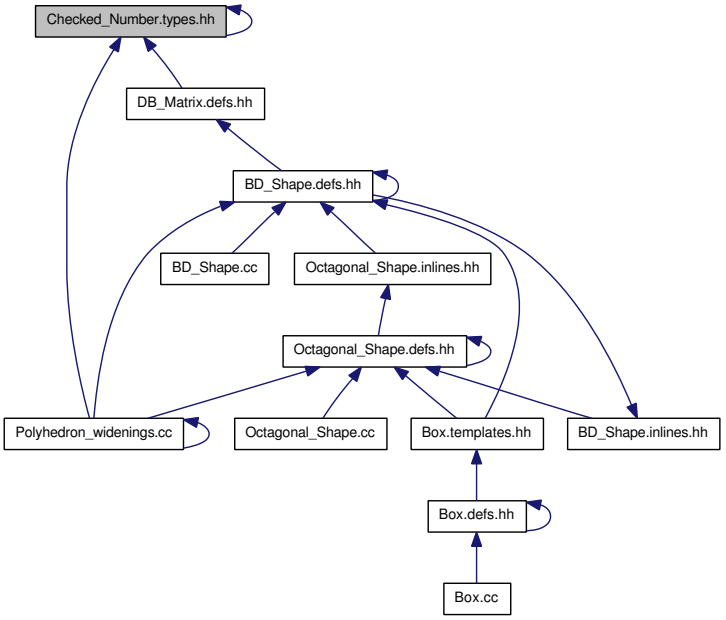
Definition at line 466 of file Checked_Number.inlines.hh.

12.38 Checked_Number.types.hh File Reference

```
#include "Coefficient_traits_template.hh"
```

Include dependency graph for Checked_Number.types.hh:

This graph shows which files directly or indirectly include this file:



Namespaces

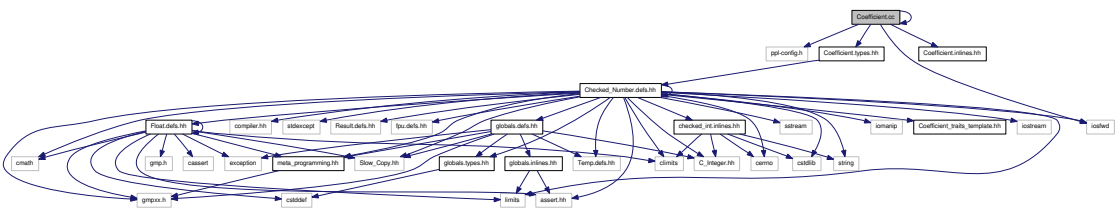
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.39 Coefficient.cc File Reference

```
#include <ppl-config.h>
#include "Coefficient.defs.hh"
#include "Coefficient.types.hh"
#include <iosfwd>
#include "Coefficient.inlines.hh"
```

Include dependency graph for Coefficient.cc:



This graph shows which files directly or indirectly include this file:

Namespaces

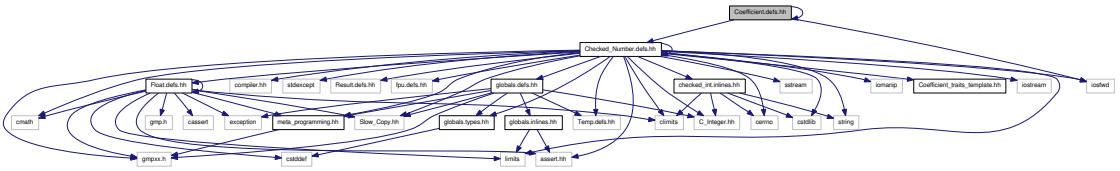
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

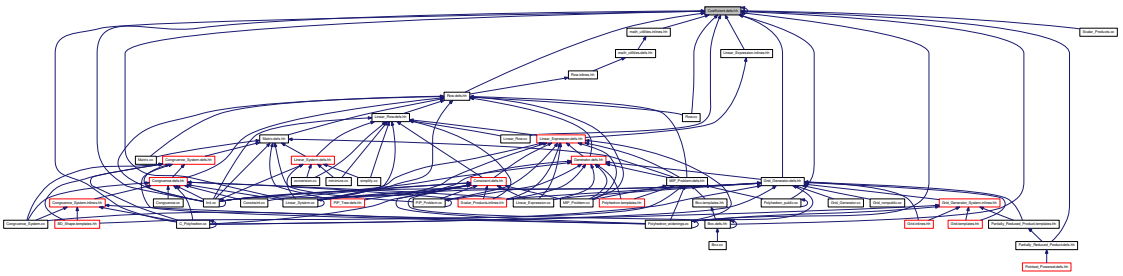
12.40 Coefficient.defs.hh File Reference

```
#include "Coefficient.types.hh"
#include "Checked_Number.defs.hh"
#include <iosfwd>
```

Include dependency graph for Coefficient.defs.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

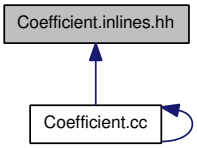
- void Parma_Polyhedra_Library::Coefficient_constants_initialize ()

  Initializes the Coefficient constants.

- void Parma_Polyhedra_Library::Coefficient_constants_finalize ()
  Finalizes the Coefficient constants.
- Coefficient_traits::const_reference Parma_Polyhedra_Library::Coefficient_zero ()
  Returns a const reference to a Coefficient with value 0.
- Coefficient_traits::const_reference Parma_Polyhedra_Library::Coefficient_one ()
  Returns a const reference to a Coefficient with value 1.

12.41 Coefficient.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



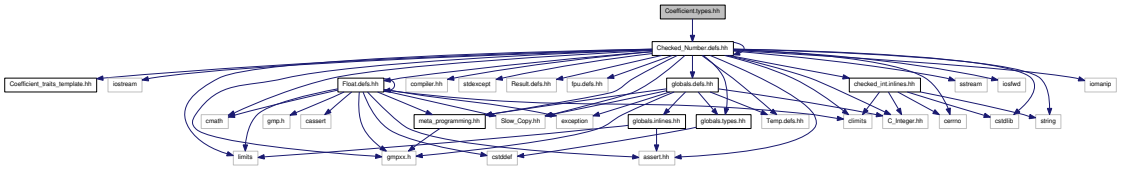
Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

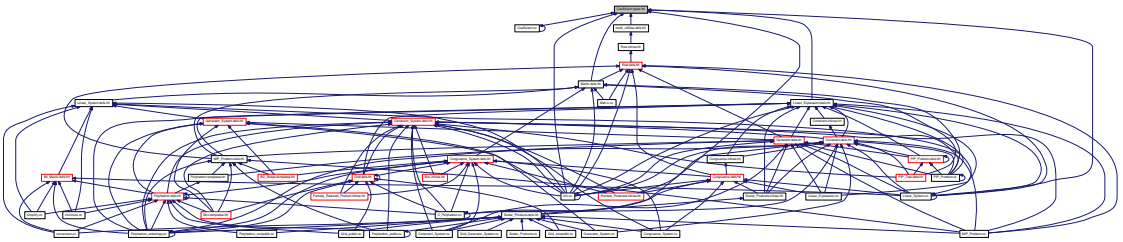
12.42 Coefficient.types.hh File Reference

```
#include "Checked_Number.defs.hh"
```

Include dependency graph for Coefficient.types.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Typedefs

- typedef PPL_COEFFICIENT_TYPE Parma_Polyhedra_Library::Coefficient

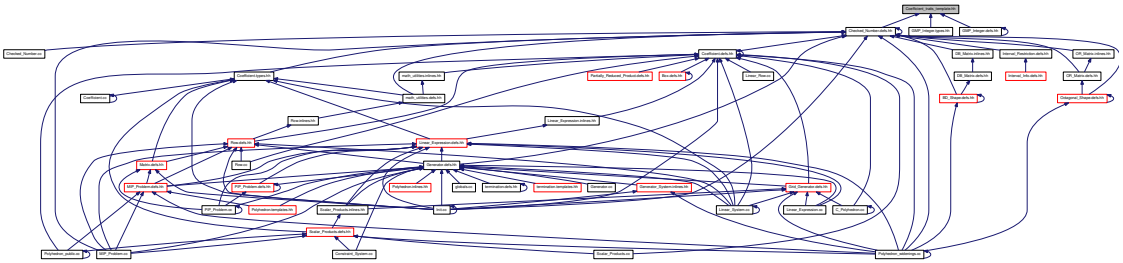
  An alias for easily naming the type of PPL coefficients.

- typedef Coefficient_traits_template< Coefficient > Parma_Polyhedra_Library::Coefficient_traits

  An alias for easily naming the coefficient traits.

12.43 Coefficient_traits_template.hh File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Coefficient_traits_template< Coefficient >

  Coefficient traits.

Namespaces

- namespace Parma_Polyhedra_Library

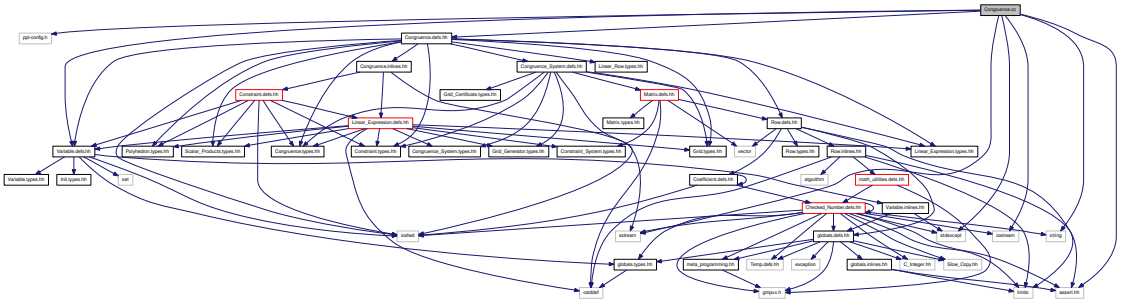
  The entire library is confined to this namespace.

12.44 Congruence.cc File Reference

```
#include <ppl-config.h>
#include "Congruence.defs.hh"
#include "Variable.defs.hh"
#include "assert.hh"
```

```
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
```

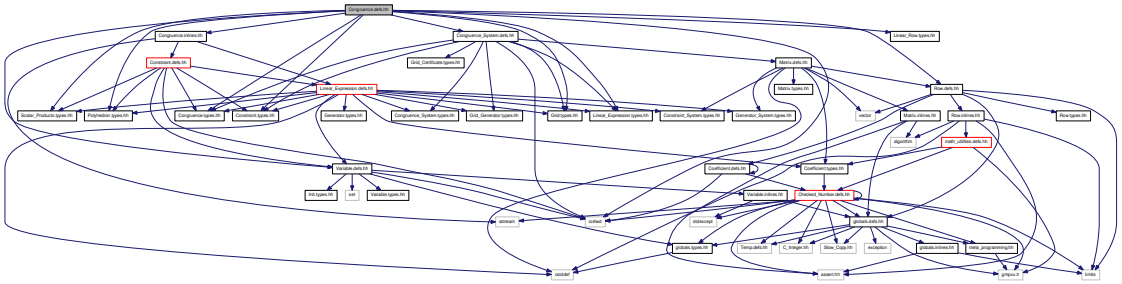
Include dependency graph for Congruence.cc:



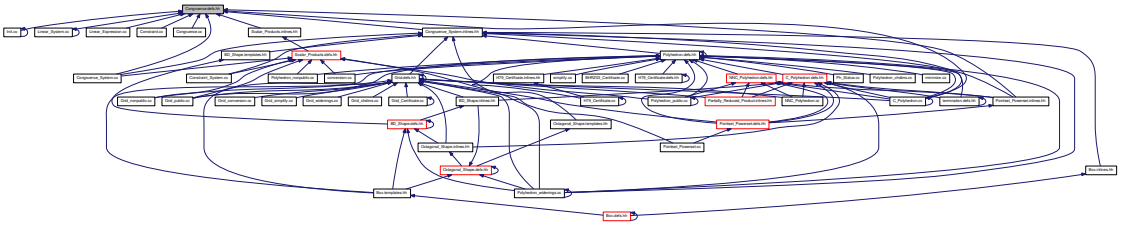
12.45 Congruence.defs.hh File Reference

```
#include "Congruence.types.hh"
#include "Scalar_Products.types.hh"
#include "Row.defs.hh"
#include "Grid.types.hh"
#include "Polyhedron.types.hh"
#include "Variable.defs.hh"
#include "Constraint.types.hh"
#include "Linear_Expression.types.hh"
#include "Linear_Row.types.hh"
#include "Congruence_System.defs.hh"
#include <iosfwd>
#include "Congruence.inlines.hh"
```

Include dependency graph for Congruence.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Congruence

  A linear congruence.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

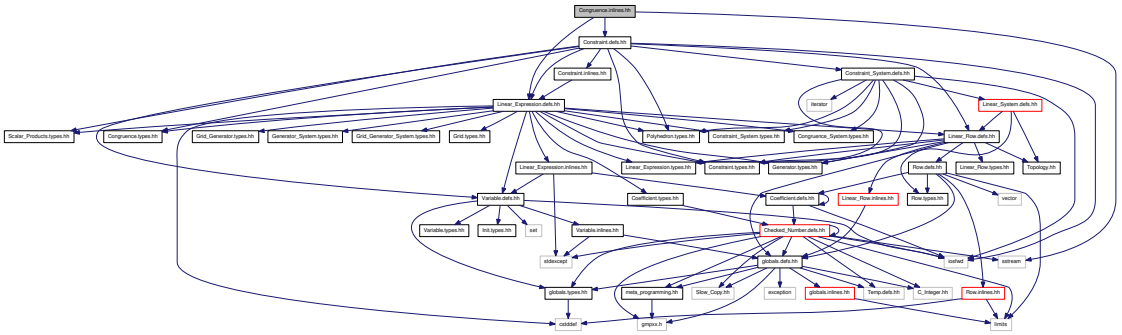
- namespace std

  The standard C++ namespace.

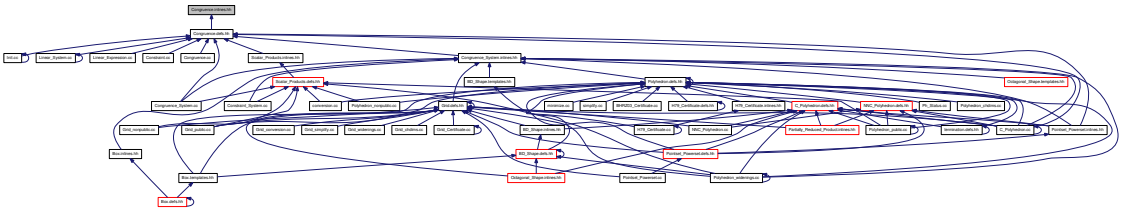
12.46 Congruence.inlines.hh File Reference

```
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include <sstream>
```

Include dependency graph for Congruence.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

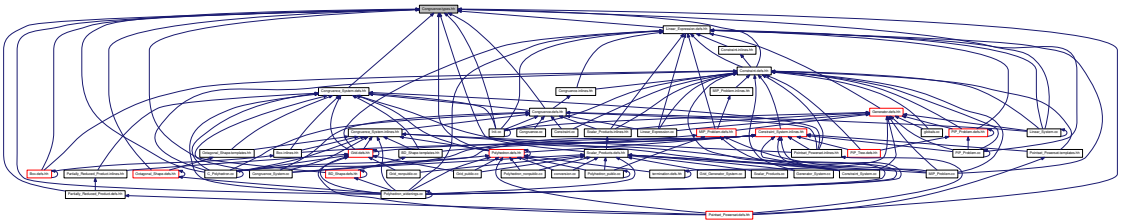
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.47 Congruence.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

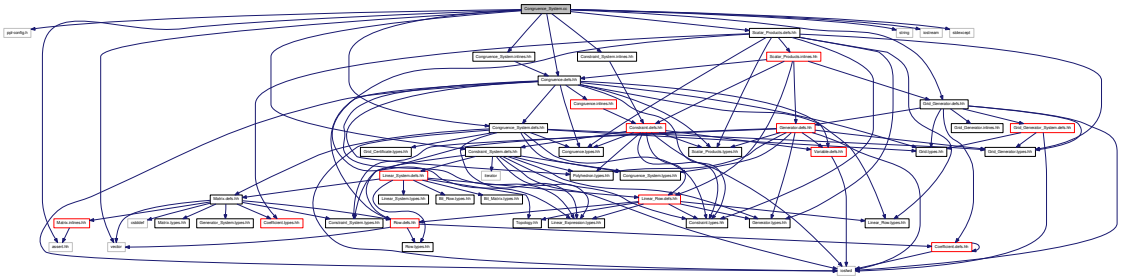
  The entire library is confined to this namespace.

12.48 Congruence_System.cc File Reference

```
#include <ppl-config.h>
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence.defs.hh"
#include "Grid_Generator.defs.hh"
```

```
#include "Scalar_Products.defs.hh"
#include "assert.hh"
#include <string>
#include <vector>
#include <iostream>
#include <stdexcept>
```

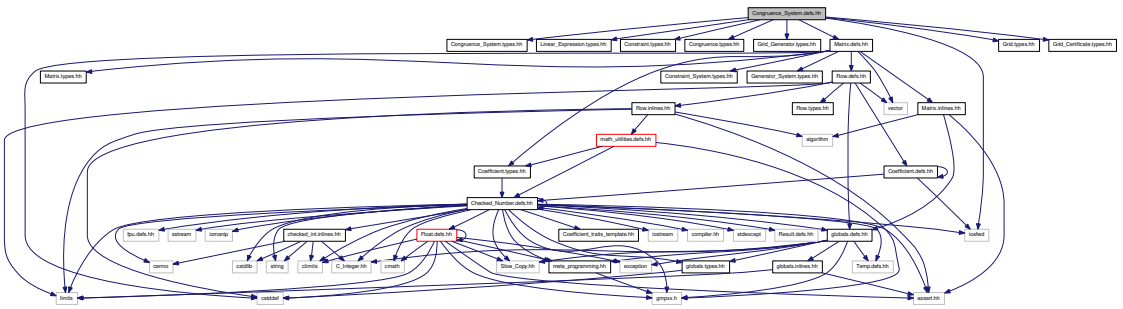
Include dependency graph for Congruence_System.cc:



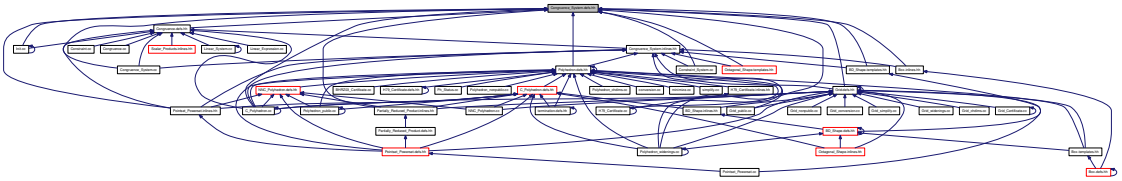
12.49 Congruence_System.defs.hh File Reference

```
#include "Congruence_System.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Congruence.types.hh"
#include "Grid_Generator.types.hh"
#include "Matrix.defs.hh"
#include "Grid.types.hh"
#include "Grid_Certificate.types.hh"
#include <iosfwd>
```

Include dependency graph for Congruence_System.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Congruence_System

  A system of congruences.

- class Parma_Polyhedra_Library::Congruence_System::const_iterator

  An iterator over a system of congruences.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

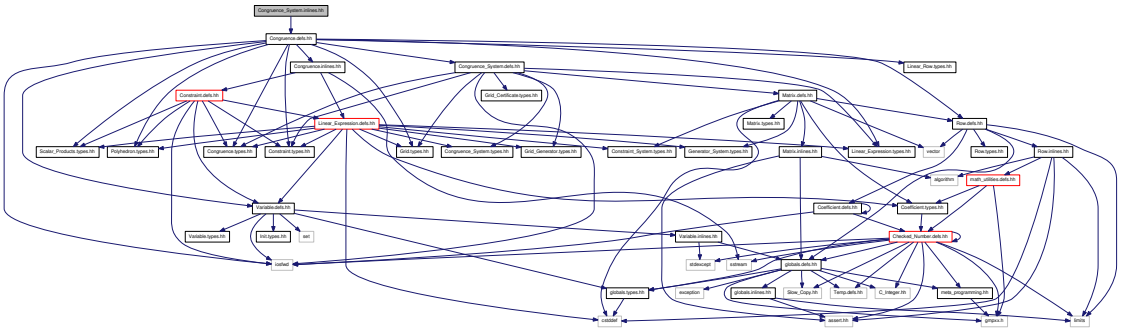
- namespace std

  The standard C++ namespace.

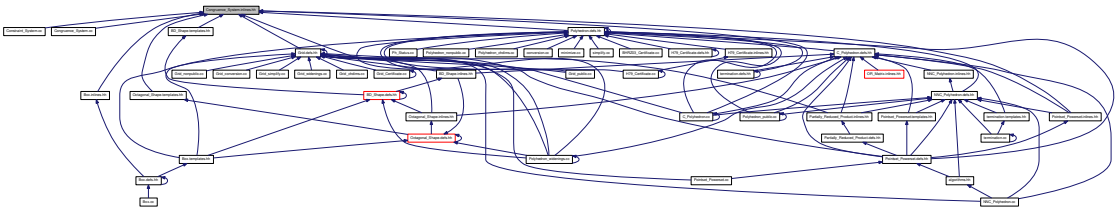
12.50 Congruence_System.inlines.hh File Reference

```
#include "Congruence.defs.hh"
```

Include dependency graph for Congruence_System.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

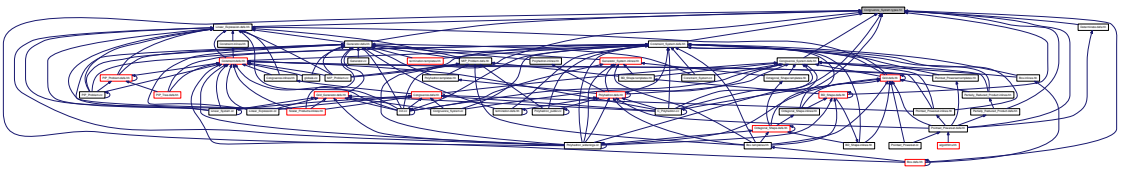
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.51 Congruence_System.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

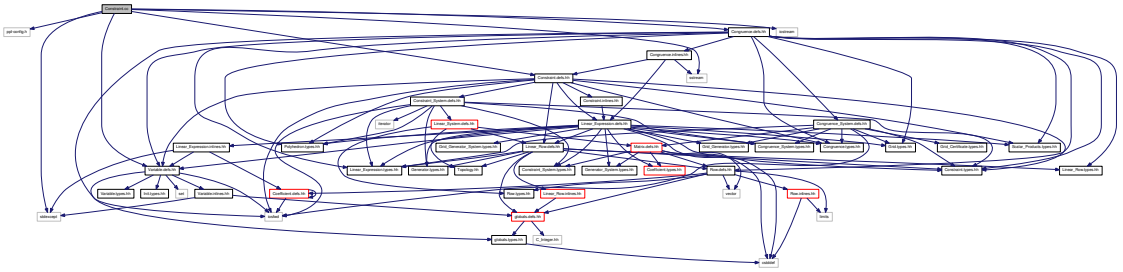
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.52 Constraint.cc File Reference

```
#include <ppl-config.h>
#include "Constraint.defs.hh"
#include "Variable.defs.hh"
#include "Congruence.defs.hh"
#include <iostream>
#include <sstream>
#include <stdexcept>
```

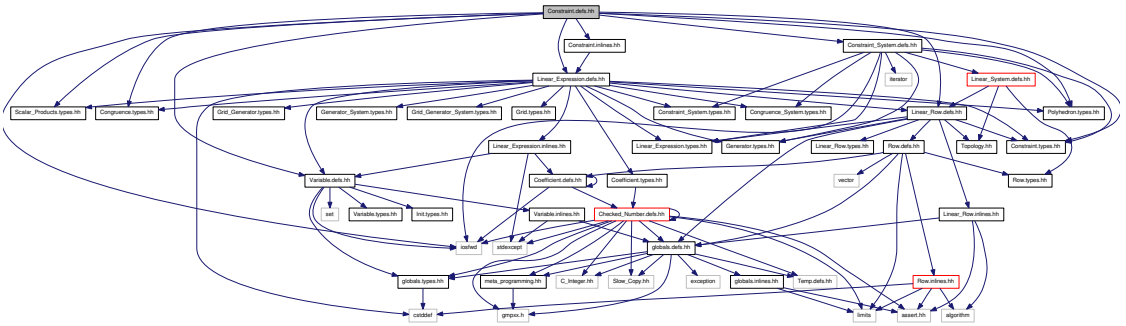
Include dependency graph for Constraint.cc:



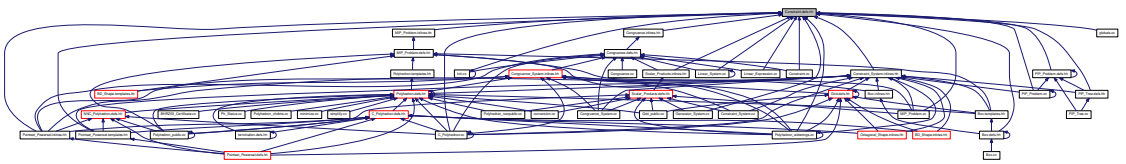
12.53 Constraint.defs.hh File Reference

```
#include "Constraint.types.hh"
#include "Scalar_Products.types.hh"
#include "Linear_Row.defs.hh"
#include "Variable.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint_System.defs.hh"
#include "Polyhedron.types.hh"
#include "Congruence.types.hh"
#include <iosfwd>
#include "Constraint.inlines.hh"
```

Include dependency graph for Constraint.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Constraint

  A linear equality or inequality.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

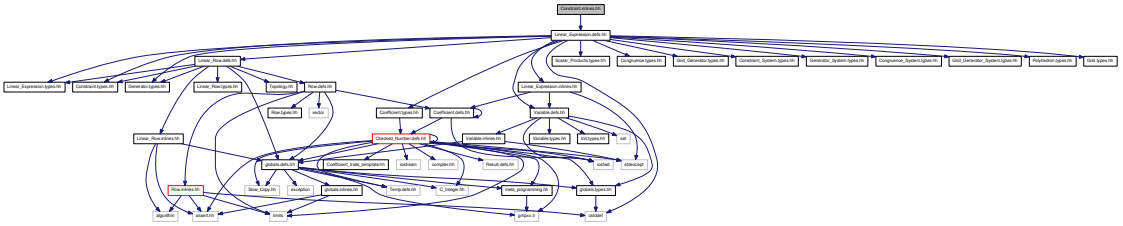
- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

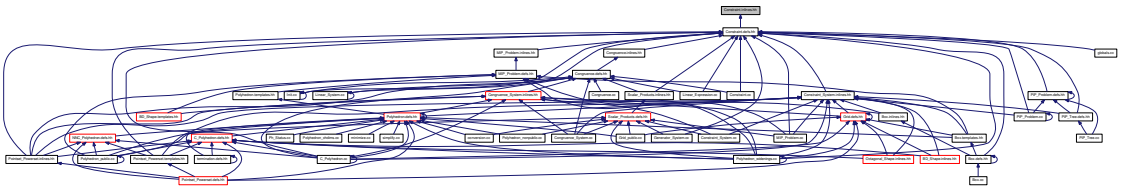
12.54 Constraint.inlines.hh File Reference

```
#include "Linear_Expression.defs.hh"
```

Include dependency graph for Constraint.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

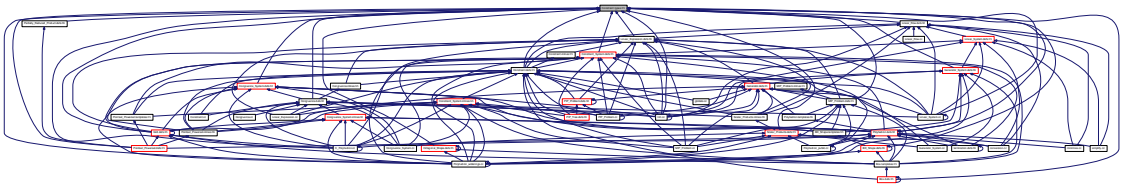
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.55 Constraint.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

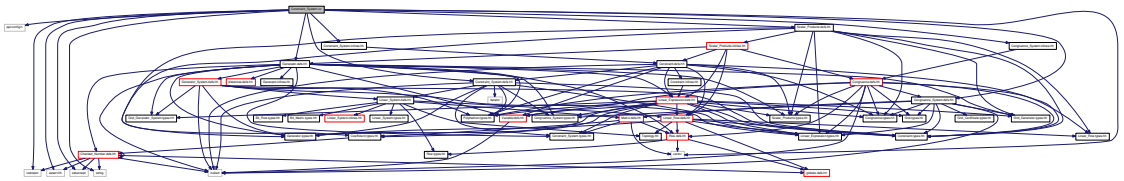
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.56 Constraint_System.cc File Reference

```
#include <ppl-config.h>
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Generator.defs.hh"
#include "Scalar_Products.defs.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "assert.hh"
#include <string>
#include <vector>
#include <iostream>
#include <stdexcept>
```

Include dependency graph for Constraint_System.cc:

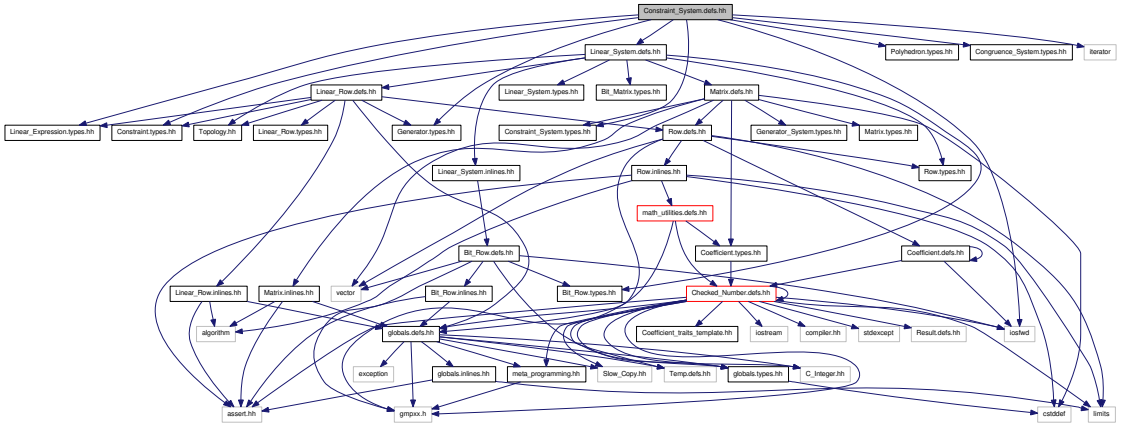


12.57 Constraint_System.defs.hh File Reference

```
#include "Constraint_System.types.hh"
#include "Linear_Expression.types.hh"
```

```
#include "Linear_System.defs.hh"
#include "Generator.types.hh"
#include "Polyhedron.types.hh"
#include "Constraint.types.hh"
#include "Congruence_System.types.hh"
#include <iterator>
#include <iosfwd>
```

Include dependency graph for Constraint_System.defs.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class Parma_Polyhedra_Library::Constraint_System

  A system of constraints.

- class Parma_Polyhedra_Library::Constraint_System::const_iterator

  An iterator over a system of constraints.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

All input/output operators are confined to this namespace.

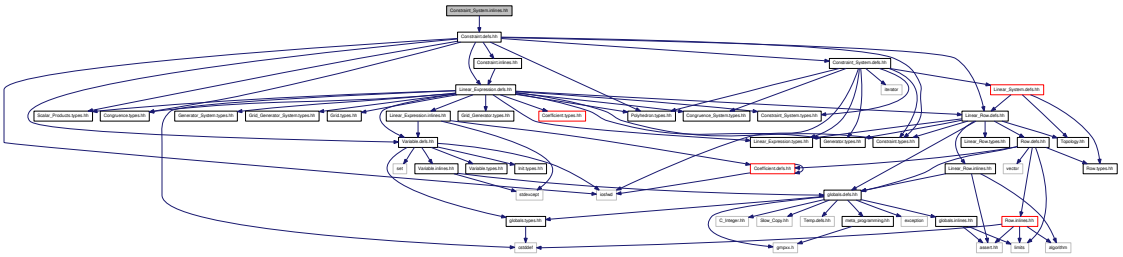
- namespace std

  The standard C++ namespace.

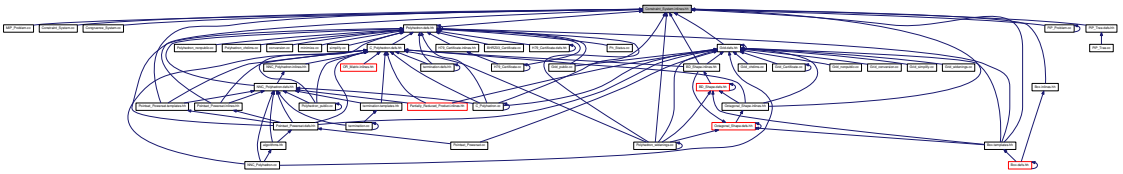
12.58 Constraint_System.inlines.hh File Reference

```
#include "Constraint.defs.hh"
```

Include dependency graph for Constraint_System.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

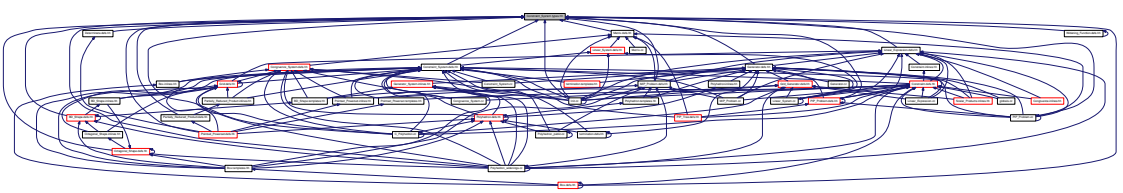
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.59 Constraint_System.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

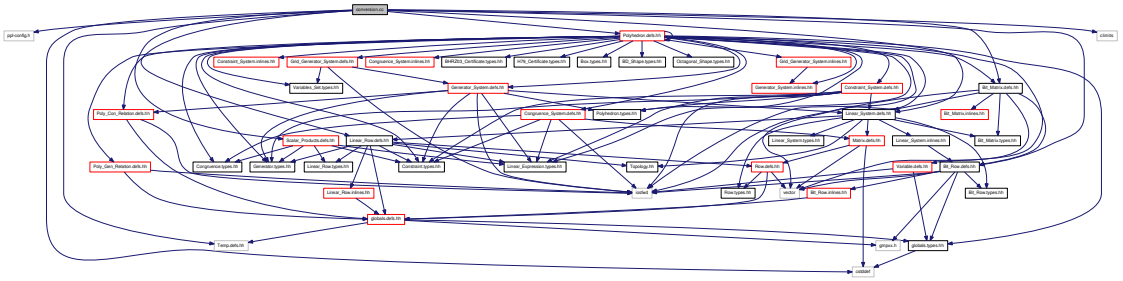
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.60 conversion.cc File Reference

```
#include <ppl-config.h>
#include "Linear_Row.defs.hh"
#include "Linear_System.defs.hh"
#include "Bit_Row.defs.hh"
#include "Bit_Matrix.defs.hh"
#include "Polyhedron.defs.hh"
#include "Scalar_Products.defs.hh"
#include "Temp.defs.hh"
#include <cstddef>
#include <climits>
```

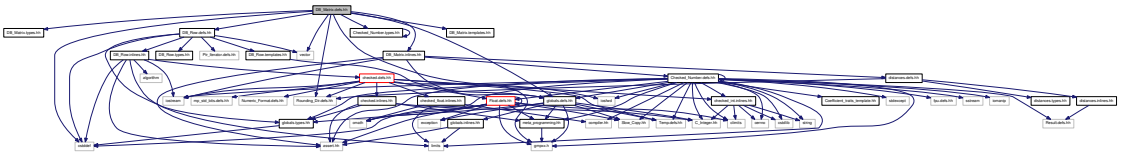
Include dependency graph for conversion.cc:



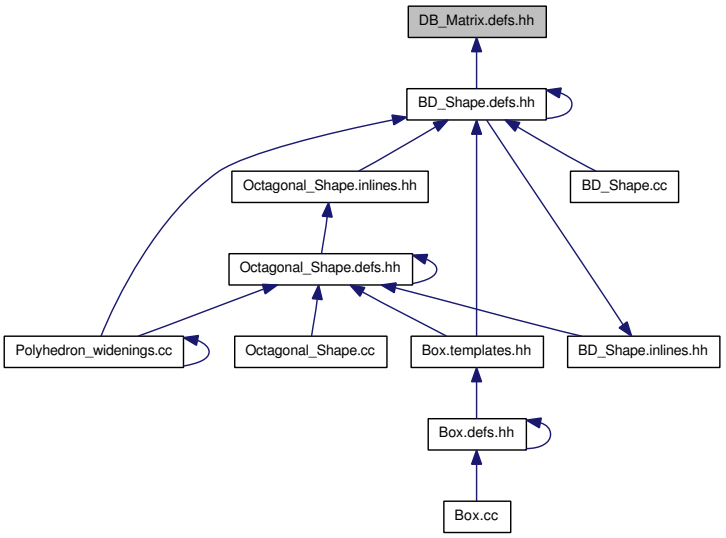
12.61 DB_Matrix.defs.hh File Reference

```
#include "DB_Matrix.types.hh"
#include "globals.defs.hh"
#include "DB_Row.defs.hh"
#include "Checked_Number.types.hh"
#include "Rounding_Dir.defs.hh"
#include <vector>
#include <cstddef>
#include <iosfwd>
#include "DB_Matrix.inlines.hh"
#include "DB_Matrix.templates.hh"
```

Include dependency graph for DB_Matrix.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::DB_Matrix< T >

  The base class for the square matrices.

- class Parma_Polyhedra_Library::DB_Matrix< T >::const_iterator

  A read-only iterator over the rows of the matrix.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

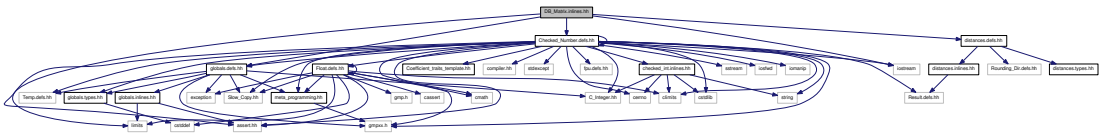
- namespace std

  The standard C++ namespace.

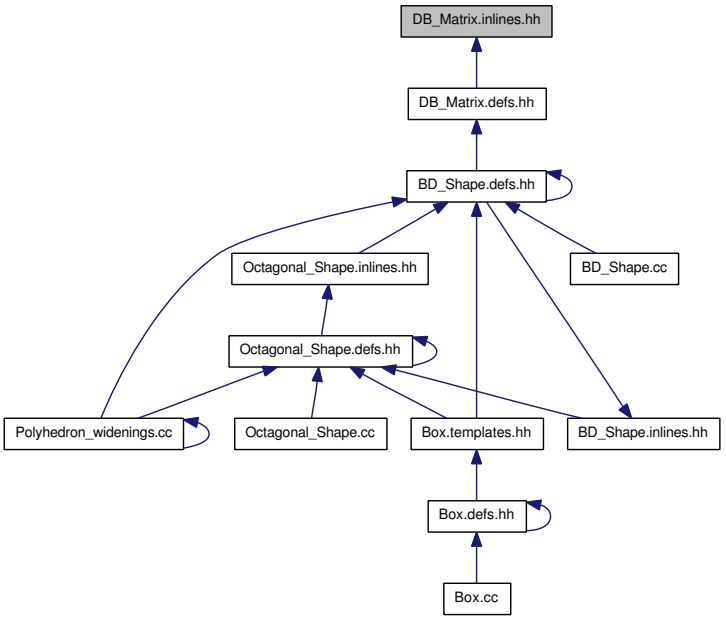
12.62 DB_Matrix.inlines.hh File Reference

```
#include "globals.defs.hh"
#include "Checked_Number.defs.hh"
#include "distances.defs.hh"
#include "assert.hh"
#include <iostream>
```

Include dependency graph for DB_Matrix.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

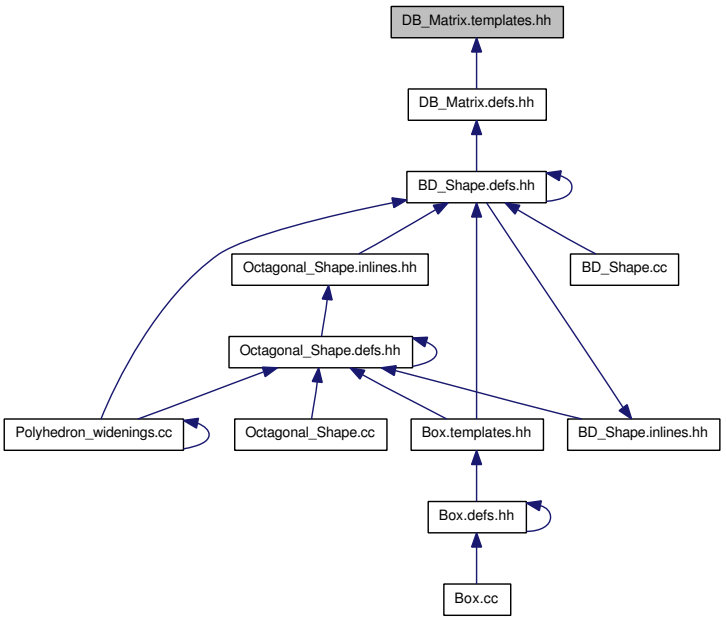
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.63 DB_Matrix.templates.hh File Reference

This graph shows which files directly or indirectly include this file:



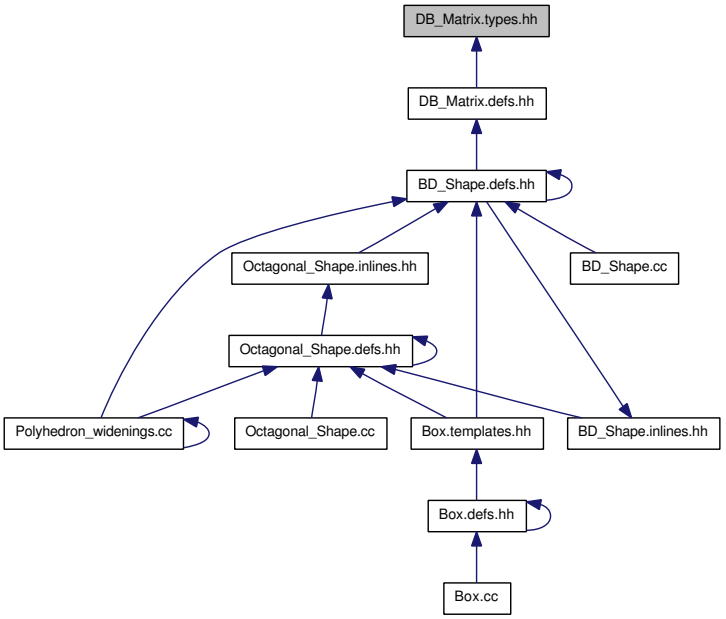
Namespaces

- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

12.64 DB_Matrix.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

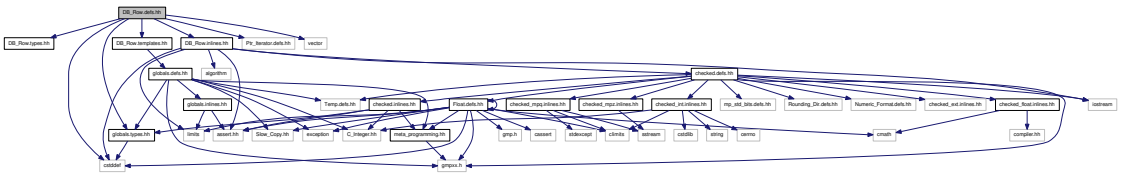
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

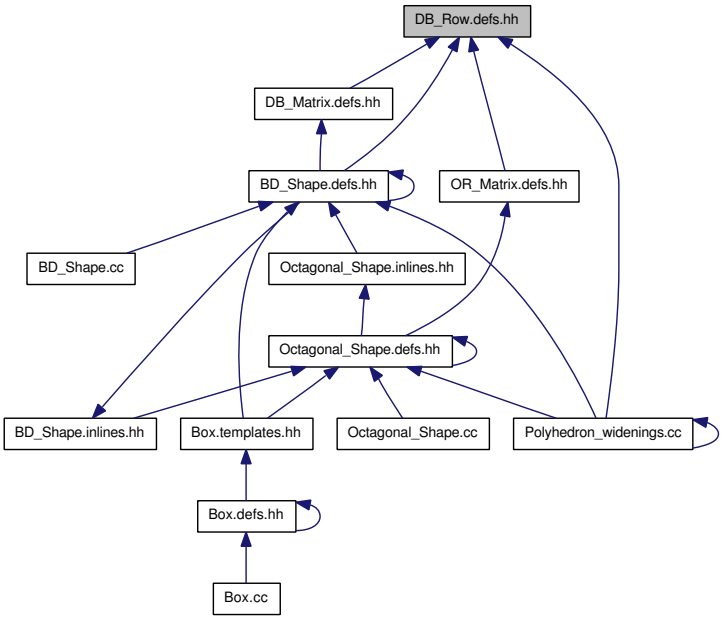
12.65 DB_Row.defs.hh File Reference

```
#include "DB_Row.types.hh"
#include "globals.types.hh"
#include "Ptr_Iterator.defs.hh"
#include <cstddef>
#include <vector>
#include "DB_Row.inlines.hh"
#include "DB_Row.templates.hh"
```

Include dependency graph for DB_Row.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >

  The handler of the actual DB_Row implementation.

- class Parma_Polyhedra_Library::DB_Row< T >

  The base class for the single rows of matrices.

- class Parma_Polyhedra_Library::DB_Row_Impl_Handler< T >::Impl

  The real implementation of a DB_Row object.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_DB_ROW_EXTRA_DEBUG 0

12.65.1 Define Documentation

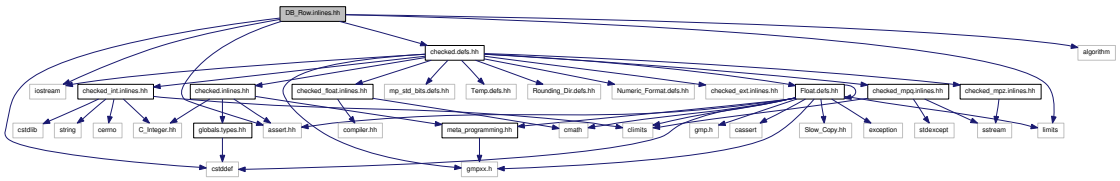
12.65.1.1 #define PPL_DB_ROW_EXTRA_DEBUG 0

Definition at line 45 of file DB_Row.defs.hh.

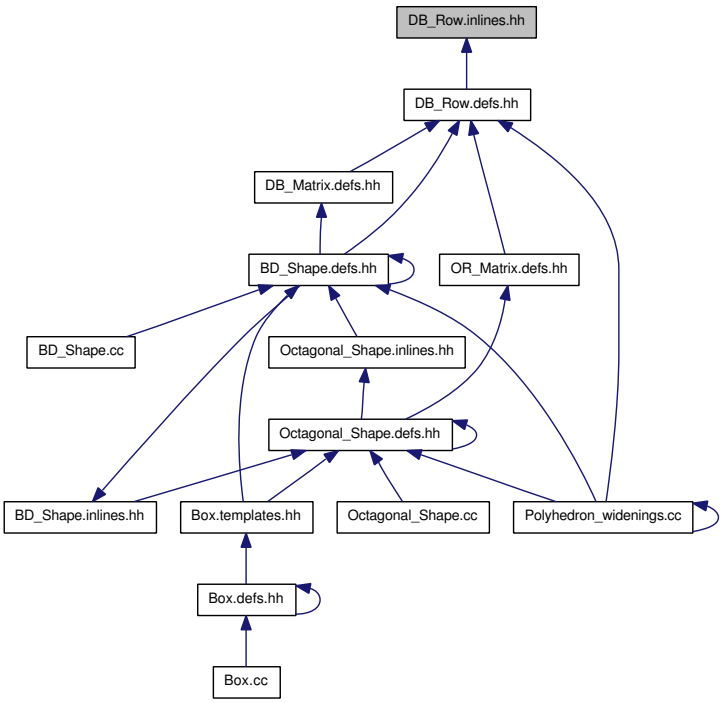
12.66 DB_Row.inlines.hh File Reference

```
#include "checked.defs.hh"
#include "assert.hh"
#include <cstddef>
#include <limits>
#include <algorithm>
#include <iostream>
```

Include dependency graph for DB_Row.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

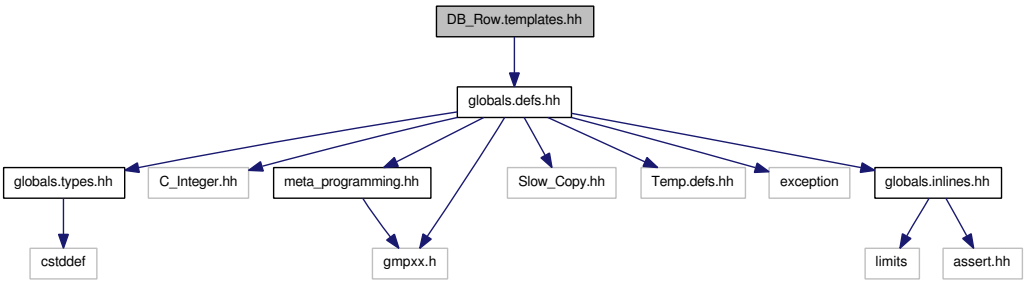
- namespace std

  The standard C++ namespace.

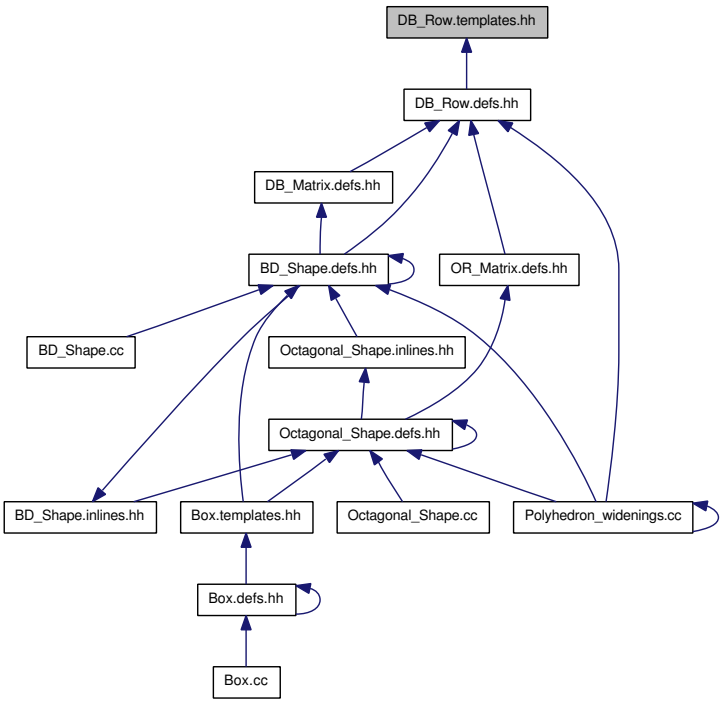
12.67 DB_Row.templates.hh File Reference

```
#include "globals.defs.hh"
```

Include dependency graph for DB_Row.templates.hh:



This graph shows which files directly or indirectly include this file:



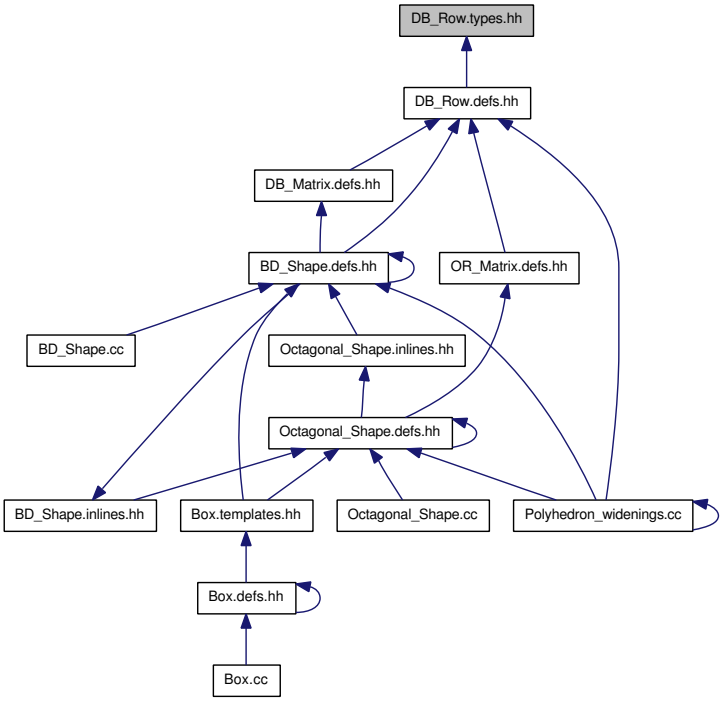
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.68 DB_Row.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

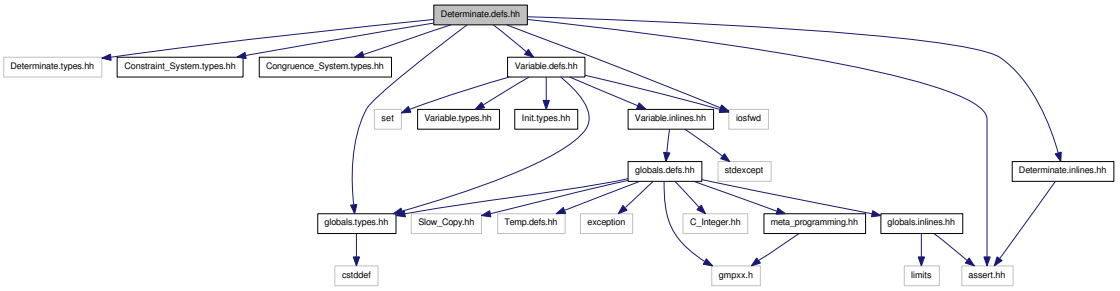
12.69 definitions.dox File Reference

12.70 Determinate.defs.hh File Reference

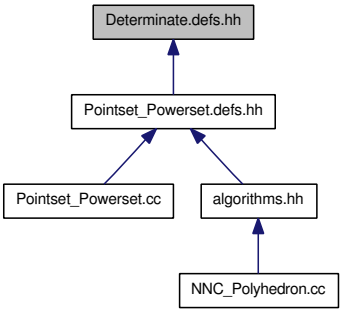
```
#include "Determinate.types.hh"
#include "Constraint_System.types.hh"
#include "Congruence_System.types.hh"
#include "Variable.defs.hh"
#include "globals.types.hh"
```

```
#include <iosfwd>
#include "assert.hh"
#include "Determinate.inlines.hh"
```

Include dependency graph for Determinate.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Determinate< PSET >

  A wrapper for PPL pointsets, providing them with a determinate constraint system interface, as defined in [Bag98].

- class Parma_Polyhedra_Library::Determinate< PSET >::Binary_Operator_Assign_Lifter< Binary_Operator_Assign >

  A function adapter for the Determinate class.

- class Parma_Polyhedra_Library::Determinate< PSET >::Rep

  The possibly shared representation of a Determinate object.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

All input/output operators are confined to this namespace.

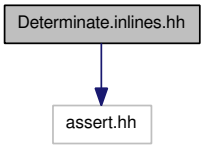
- namespace std

  The standard C++ namespace.

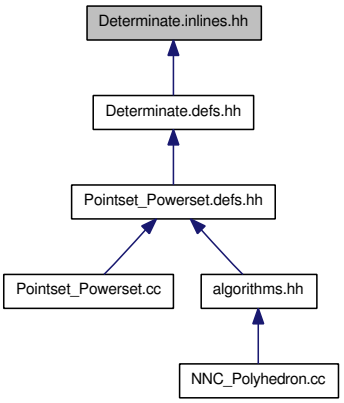
12.71 Determinate.inlines.hh File Reference

```
#include "assert.hh"
```

Include dependency graph for Determinate.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

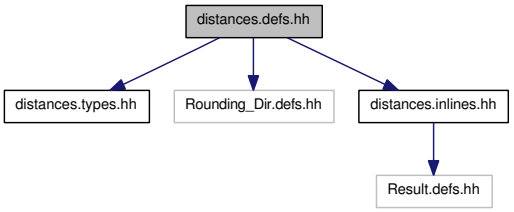
  The standard C++ namespace.

12.72 distances.defs.hh File Reference

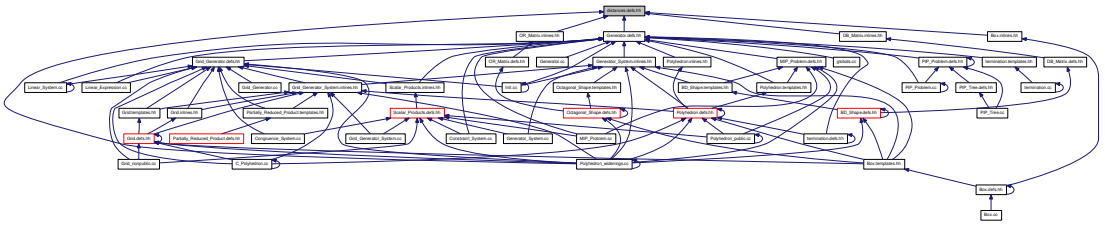
```
#include "distances.types.hh"
```

```
#include "Rounding_Dir.defs.hh"
#include "distances.inlines.hh"
```

Include dependency graph for distances.defs.hh:



This graph shows which files directly or indirectly include this file:



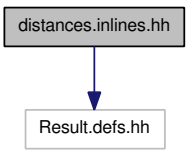
Classes

- struct Parma_Polyhedra_Library::Rectilinear_Distance_Specialization< Temp >
- struct Parma_Polyhedra_Library::Euclidean_Distance_Specialization< Temp >
- struct Parma_Polyhedra_Library::L_Infinity_Distance_Specialization< Temp >

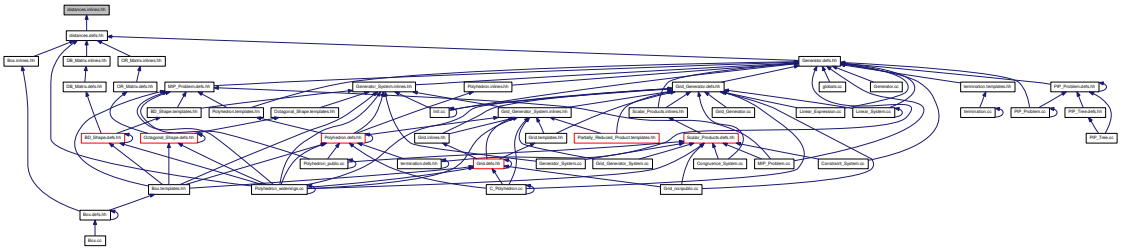
12.73 distances.inlines.hh File Reference

```
#include "Result.defs.hh"
```

Include dependency graph for distances.inlines.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::maybe_assign_struct< To, From >
- struct Parma_Polyhedra_Library::maybe_assign_struct< Type, Type >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

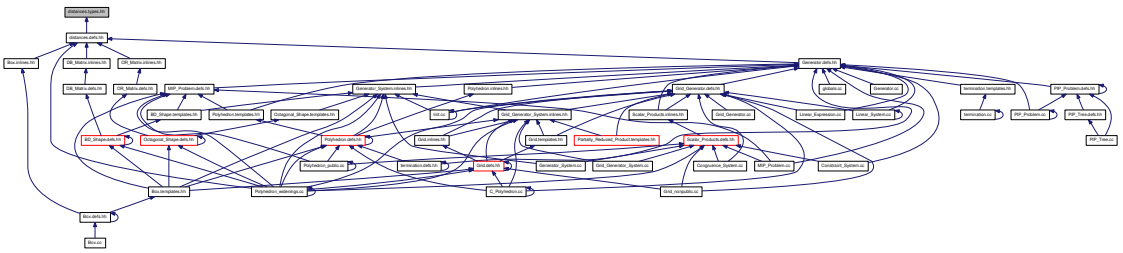
Functions

- template<typename To , typename From >
  Result Parma_Polyhedra_Library::maybe_assign (const To *&top, To &tmp, const From &from, Rounding_Dir dir)

  Assigns `top` *a pointer to a location that holds the conversion, according to* `dir`*, of* `from` *to type* `To`*. When necessary, and only when necessary, the variable* `tmp` *is used to hold the result of conversion.*

12.74 distances.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

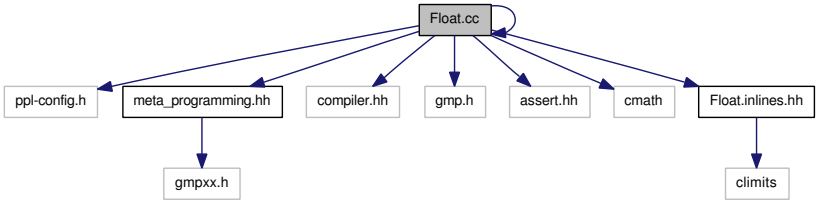
  The entire library is confined to this namespace.

12.75 fdl.dox File Reference

12.76 Float.cc File Reference

```
#include <ppl-config.h>
#include "Float.defs.hh"
#include "meta_programming.hh"
#include "compiler.hh"
#include <gmp.h>
#include "assert.hh"
#include <cmath>
#include "Float.inlines.hh"
```

Include dependency graph for Float.cc:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace Parma_Polyhedra_Library

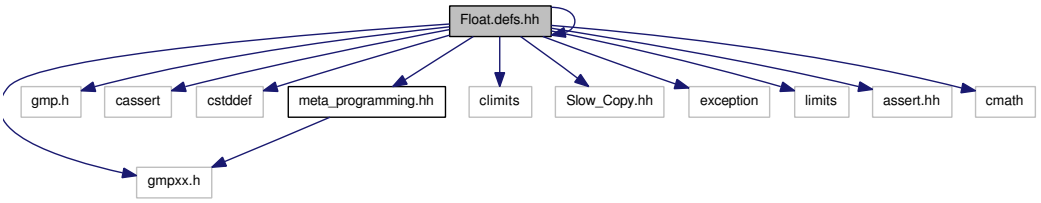
  The entire library is confined to this namespace.

12.77 Float.defs.hh File Reference

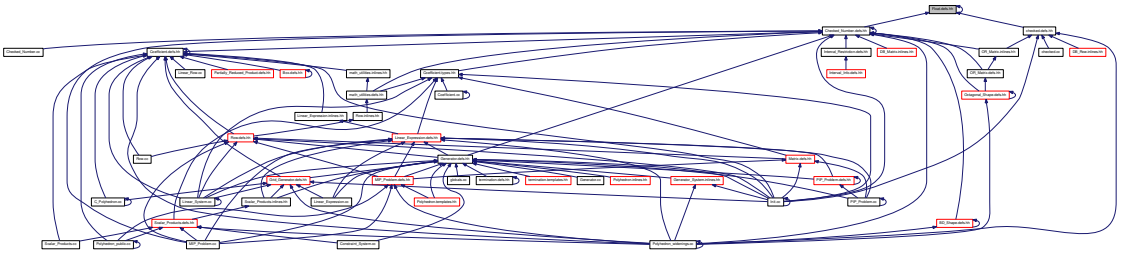
```
#include "meta_programming.hh"
#include <gmpxx.h>
#include <gmp.h>
#include <cassert>
#include <cstddef>
#include "meta_programming.hh"
#include <climits>
#include "Slow_Copy.hh"
```

```
#include <exception>
#include <limits>
#include "assert.hh"
#include <cmath>
```

Include dependency graph for Float.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::float_ieee754_half
- struct Parma_Polyhedra_Library::float_ieee754_single
- struct Parma_Polyhedra_Library::float_ieee754_double
- struct Parma_Polyhedra_Library::float_ibm_single
- struct Parma_Polyhedra_Library::float_ibm_double
- struct Parma_Polyhedra_Library::float_intel_double_extended
- struct Parma_Polyhedra_Library::float_ieee754_quad
- class Parma_Polyhedra_Library::Float< T >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Defines

- #define PPL_NAN (HUGE_VAL - HUGE_VAL)

12.77.1 Define Documentation

12.77.1.1 #define PPL_NAN (HUGE_VAL - HUGE_VAL)

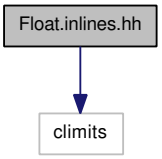
Definition at line 36 of file Float.defs.hh.

Referenced by Parma_Polyhedra_Library::Checked::assign_special_float().

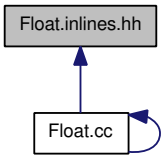
12.78 Float.inlines.hh File Reference

```
#include <climits>
```

Include dependency graph for Float.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

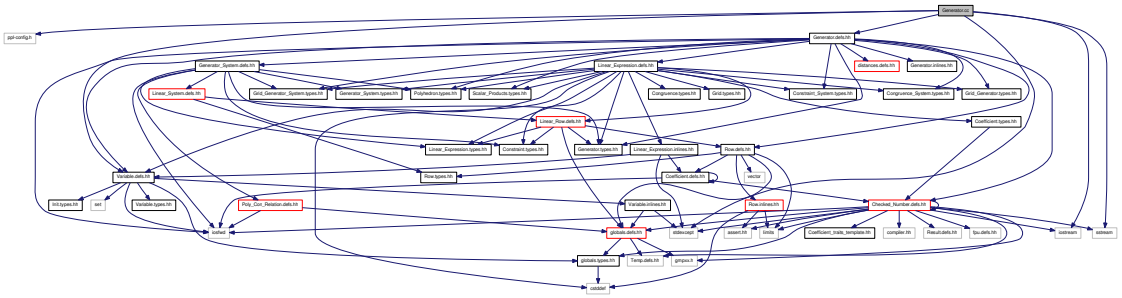
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.79 Generator.cc File Reference

```
#include <ppl-config.h>
#include "Generator.defs.hh"
#include "Variable.defs.hh"
#include <iostream>
#include <sstream>
#include <stdexcept>
```

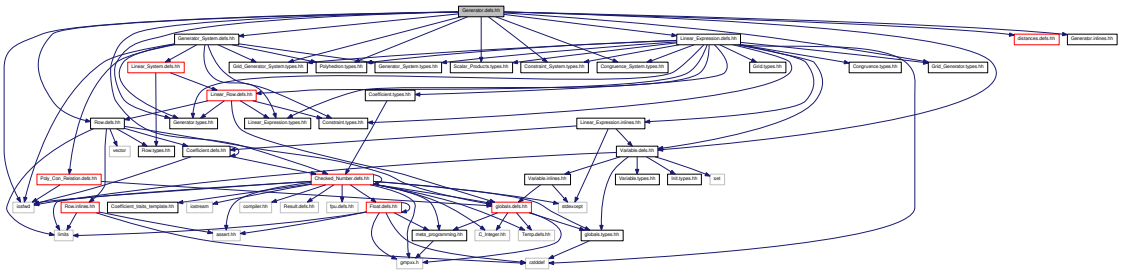
Include dependency graph for Generator.cc:



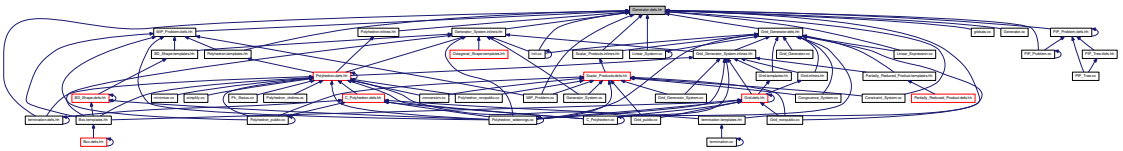
12.80 Generator.defs.hh File Reference

```
#include "Generator.types.hh"
#include "Scalar_Products.types.hh"
#include "Row.defs.hh"
#include "Variable.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.defs.hh"
#include "Congruence_System.types.hh"
#include "Linear_Expression.defs.hh"
#include "Polyhedron.types.hh"
#include "Grid_Generator.types.hh"
#include "Grid_Generator_System.types.hh"
#include "Checked_Number.defs.hh"
#include "distances.defs.hh"
#include <iosfwd>
#include "Generator.inlines.hh"
```

Include dependency graph for Generator.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Generator

  A line, ray, point or closure point.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

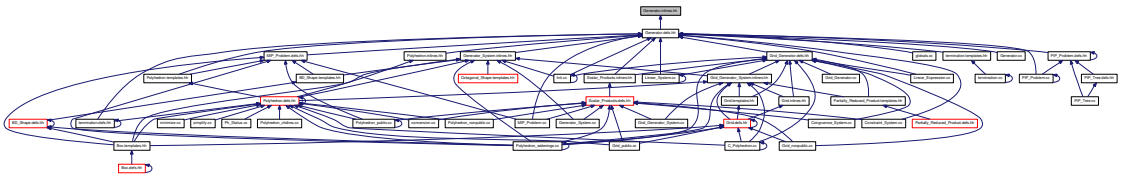
  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

12.81 Generator.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

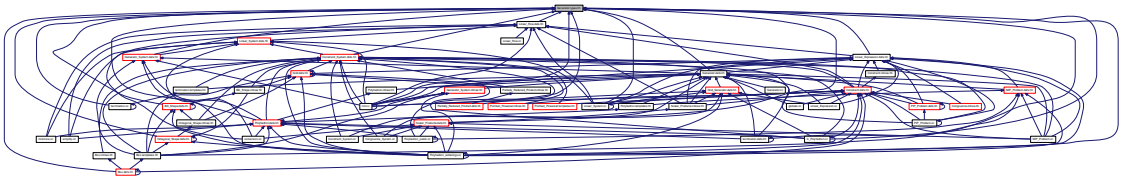
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.82 Generator.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

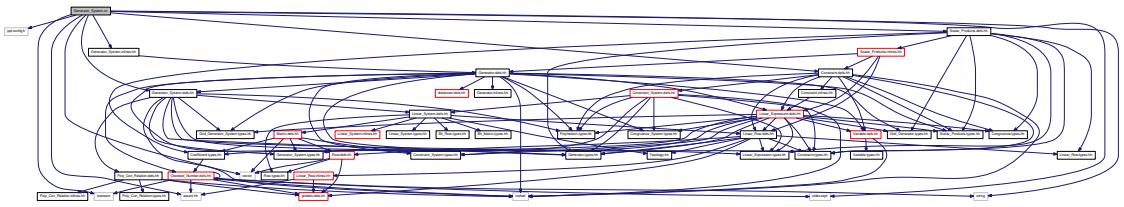
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.83 Generator_System.cc File Reference

```
#include <ppl-config.h>
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Constraint.defs.hh"
#include "Scalar_Products.defs.hh"
#include "assert.hh"
#include <string>
#include <vector>
#include <iostream>
#include <stdexcept>
```

Include dependency graph for Generator_System.cc:

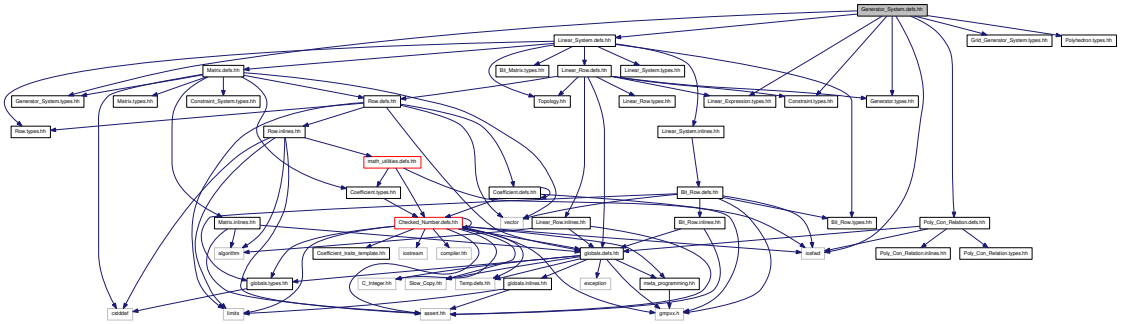


12.84 Generator_System.defs.hh File Reference

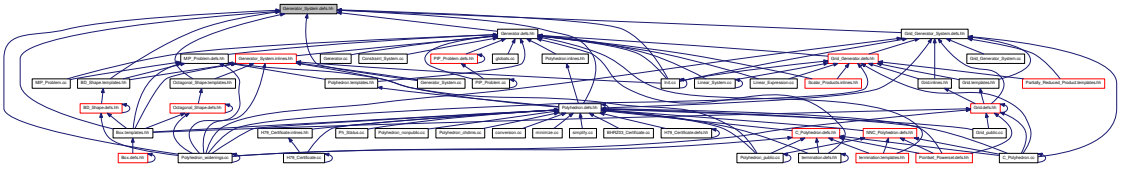
```
#include "Generator_System.types.hh"
#include "Grid_Generator_System.types.hh"
#include "Linear_Expression.types.hh"
```

```
#include "Linear_System.defs.hh"
#include "Generator.types.hh"
#include "Constraint.types.hh"
#include "Polyhedron.types.hh"
#include "Poly_Con_Relation.defs.hh"
#include <iosfwd>
```

Include dependency graph for Generator_System.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Generator_System

  A system of generators.

- class Parma_Polyhedra_Library::Generator_System::const_iterator

  An iterator over a system of generators.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

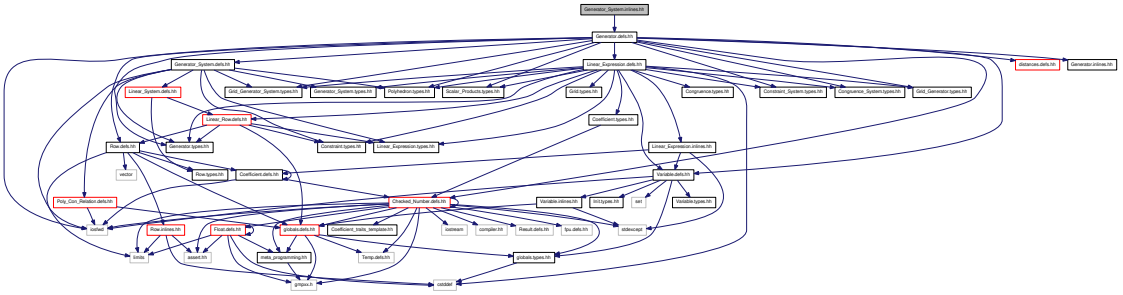
- namespace std

  The standard C++ namespace.

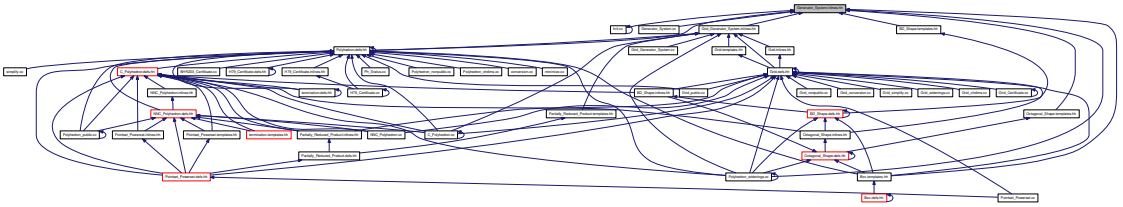
12.85 Generator_System.inlines.hh File Reference

```
#include "Generator.defs.hh"
```

Include dependency graph for Generator_System.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

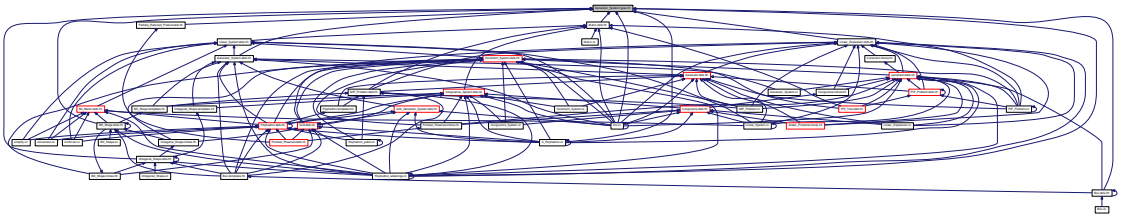
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.86 Generator_System.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

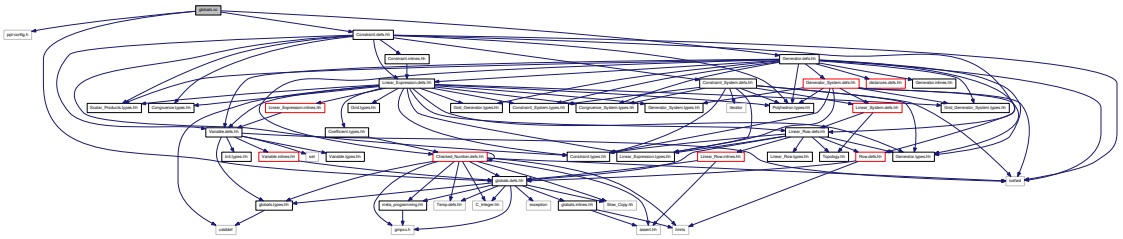
- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

12.87 globals.cc File Reference

```
#include <ppl-config.h>
#include "globals.defs.hh"
#include "Constraint.defs.hh"
#include "Generator.defs.hh"
```

Include dependency graph for globals.cc:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

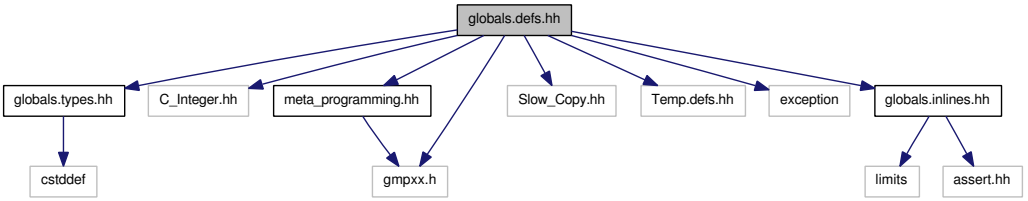
- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

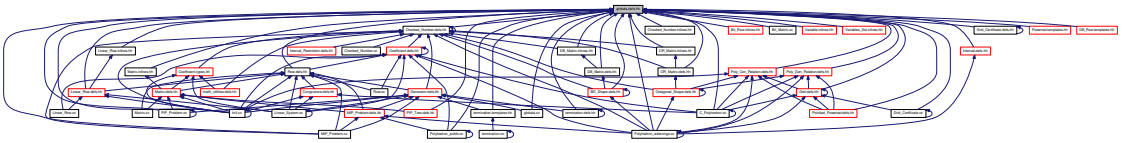
12.88 globals.defs.hh File Reference

```
#include "globals.types.hh"
#include "C_Integer.hh"
#include "meta_programming.hh"
#include "Slow_Copy.hh"
#include "Temp.defs.hh"
#include <exception>
#include <gmpxx.h>
#include "globals.inlines.hh"
```

Include dependency graph for globals.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Weightwatch_Traits

  Traits class for the deterministic timeout mechanism.

- class Parma_Polyhedra_Library::Throwable

  User objects the PPL can throw.

- struct Parma_Polyhedra_Library::Recycle_Input

  A tag class.

- struct Parma_Polyhedra_Library::Fit< T, v, Enable >
- struct Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type >
- struct Parma_Polyhedra_Library::TConstant< T, v >
- struct Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, Enable >
- struct Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed||!Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value))>::type >
- struct Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(!prefer_signed||!Fit< typename C_Integer< T >::smaller_signed_type, v >::value))>::type >
- struct Parma_Polyhedra_Library::Constant< v, prefer_signed >
- struct Parma_Polyhedra_Library::Has_OK< T, Enable >
- struct Parma_Polyhedra_Library::Has_OK< T, typename Enable_If_Is< bool(T::*)() const,&T::OK >::type >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

Defines

- #define PPL_DIRTY_TEMP_COEFFICIENT(id) PPL_DIRTY_TEMP0(Coefficient, id)

  Declare a local variable named `id`, of type Coefficient, and containing an unknown initial value.

- #define PPL_PROFILE_ADD_WEIGHT 0
- #define WEIGHT_BEGIN()
- #define WEIGHT_ADD(delta)
- #define WEIGHT_ADD_MUL(delta, factor)
- #define PPL_STR(s) #s
- #define PPL_XSTR(s) PPL_STR(s)
- #define PPL_OUTPUT_DECLARATIONS
- #define PPL_OUTPUT_DEFINITIONS(class_name)
- #define PPL_OUTPUT_DEFINITIONS_ASCII_ONLY(class_name)
- #define PPL_OUTPUT_TEMPLATE_DEFINITIONS(type_symbol, class_prefix)
- #define PPL_OUTPUT_2_PARAM_TEMPLATE_DEFINITIONS(type_symbol1,type_-symbol2,class_prefix)
- #define PPL_OUTPUT_3_PARAM_TEMPLATE_DEFINITIONS(type_symbol1,type_-symbol2,type_symbol3,class_prefix)
- #define PPL_OUTPUT_TEMPLATE_DEFINITIONS_ASCII_ONLY(type_symbol, class_prefix)
- #define FOK(T) inline bool f_OK(const T&) { return true; }

Functions

- dimension_type Parma_Polyhedra_Library::not_a_dimension ()

  Returns a value that does not designate a valid dimension.

- template<typename T >
  Enable_If< Slow_Copy< T >::value, void >::type Parma_Polyhedra_Library::swap (T &, T &)

  Make sure swap() is specialized when needed.

- dimension_type Parma_Polyhedra_Library::compute_capacity (dimension_type requested_size, dimension_type maximum_size)

  Speculative allocation function.

- template<typename T >
  Enable_If< Has_OK< T >::value, bool >::type Parma_Polyhedra_Library::f_OK (const T &to)

Memory Size Inspection Functions

- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_-Library::total_memory_in_bytes (const T &)

  For native types, returns the total size in bytes of the memory occupied by the type of the (unused) parameter, i.e., 0.

- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_Library::external_memory_in_bytes (const T &)
  For native types, returns the size in bytes of the memory managed by the type of the (unused) parameter, i.e., 0.

- memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpz_class &x)
  Returns the total size in bytes of the memory occupied by x.

- memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpz_class &x)
  Returns the size in bytes of the memory managed by x.

- memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpq_class &x)
  Returns the total size in bytes of the memory occupied by x.

- memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpq_class &x)
  Returns the size in bytes of the memory managed by x.

Variables

- unsigned int Parma_Polyhedra_Library::Implementation::in_assert = 0
  Non zero during evaluation of PPL_ASSERT expression.

- const Throwable *volatile Parma_Polyhedra_Library::abandon_expensive_computations = 0
  A pointer to an exception object.

12.88.1 Define Documentation

12.88.1.1 #define FOK(T) inline bool f_OK(const T&) { return true; }

Definition at line 448 of file globals.defs.hh.

12.88.1.2 #define PPL_DIRTY_TEMP_COEFFICIENT(id) PPL_DIRTY_TEMP0(Coefficient, id)

Declare a local variable named `id`, of type Coefficient, and containing an unknown initial value.

Use of this macro to declare temporaries of type Coefficient results in decreased memory allocation overhead and in better locality.

Definition at line 62 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::Octagonal_Shape< T >::add_constraint(), Parma_Polyhedra_Library::BD_Shape< T >::add_constraint(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_image(), Parma_Polyhedra_Library::Grid_Generator_System::affine_image(), Parma_Polyhedra_Library::Generator_System::affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::affine_preimage(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::approximate_partition_aux(),

Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter::Artificial_Parameter(), Parma_Polyhedra_Library::BD_Shape< T >::BD_Shape(), Parma_Polyhedra_Library::BD_Shape< T >::BFT00_upper_bound_assign_if_exact(), Parma_Polyhedra_Library::Polyhedron::BHRZ03_evolving_rays(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_image(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::bounded_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::bounded_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::bounds(), Parma_Polyhedra_Library::BD_Shape< T >::bounds(), Parma_Polyhedra_Library::MIP_Problem::choose_branching_variable(), Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_generator(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::Octagonal_Shape< T >::constraints(), Parma_Polyhedra_Library::BD_Shape< T >::constraints(), Parma_Polyhedra_Library::Polyhedron::contains_integer_point(), Parma_Polyhedra_Library::Grid::conversion(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_minus_v_pm_u_bounds(), Parma_Polyhedra_Library::Octagonal_Shape< T >::deduce_v_pm_u_bounds(), Parma_Polyhedra_Library::Polyhedron::drop_some_non_integer_points(), Parma_Polyhedra_Library::Polyhedron::frequency(), Parma_Polyhedra_Library::Octagonal_Shape< T >::frequency(), Parma_Polyhedra_Library::Box< ITV >::frequency(), Parma_Polyhedra_Library::BD_Shape< T >::frequency(), Parma_Polyhedra_Library::Grid::frequency_no_check(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Grid::generalized_affine_image(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_image(), Parma_Polyhedra_Library::Polyhedron::generalized_affine_preimage(), Parma_Polyhedra_Library::Octagonal_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::Grid::generalized_affine_preimage(), Parma_Polyhedra_Library::Box< ITV >::generalized_affine_preimage(), Parma_Polyhedra_Library::BD_Shape< T >::generalized_affine_preimage(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index(), Parma_Polyhedra_Library::Octagonal_Shape< T >::get_limiting_octagon(), Parma_Polyhedra_Library::BD_Shape< T >::get_limiting_shape(), Parma_Polyhedra_Library::Grid::Grid(), Parma_Polyhedra_Library::Scalar_Products::homogeneous_sign(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot(), Parma_Polyhedra_Library::Generator::is_matching_closure_point(), Parma_Polyhedra_Library::MIP_Problem::is_mip_satisfiable(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), Parma_Polyhedra_Library::Linear_Row::linear_combine(), Parma_Polyhedra_Library::Polyhedron::max_min(), Parma_Polyhedra_Library::Octagonal_Shape< T >::max_min(), Parma_Polyhedra_Library::Grid::max_min(), Parma_Polyhedra_Library::BD_Shape< T >::max_min(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::maximize(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::maximize(), Parma_Polyhedra_Library::Pointset_Powerset< PSET >::minimize(), Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >::minimize(), Parma_Polyhedra_Library::Octagonal_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_congruences(), Parma_Polyhedra_Library::BD_Shape< T >::minimized_constraints(), Parma_Polyhedra_Library::Row::normalize(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize(), Parma_Polyhedra_Library::normalize2(), Parma_Polyhedra_Library::Grid::normalize_divisors(), Parma_Polyhedra_Library::Congruence_System::normalize_moduli(), Parma_Polyhedra_Library::Octagonal_Shape< T >::Octagonal_Shape(), Parma_Polyhedra_Library::MIP_Problem::OK(), Parma_Polyhedra_Library::Linear_Expression::operator<<(), Parma_Polyhedra_Library::Grid_Generator::operator<<(), Parma_Polyhedra_Library::Generator::operator<<(), Parma_Polyhedra_Library::Constraint::operator<<(), Parma_Polyhedra_Library::Congruence::operator<<(), Parma_Polyhedra_Library::Matrix::permute_columns(), Parma_Polyhedra_Library::Polyhedron::Polyhedron(), Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >::product_reduce(), Parma_Polyhedra_Library::Grid::reduce_congruence_with_equality(), Parma_Polyhedra_Library::Grid::reduce_equality_with_equality(), Parma_Polyhedra_Library::Grid::reduce_line_with_line(), Parma_Polyhedra_Library::Grid::reduce_parameter_with_line(), Parma_Polyhedra_Library::Grid::reduce_pc_with_pc(), Parma_Polyhedra_

Library::Grid::reduce_reduced(), Parma_Polyhedra_Library::Scalar_Products::reduced_sign(), Parma_-Polyhedra_Library::Octagonal_Shape< T >::refine(), Parma_Polyhedra_Library::BD_Shape< T >::refine(), Parma_Polyhedra_Library::Octagonal_Shape< T >::refine_no_check(), Parma_Polyhedra_-Library::BD_Shape< T >::refine_no_check(), Parma_Polyhedra_Library::Polyhedron::relation_-with(), Parma_Polyhedra_Library::Octagonal_Shape< T >::relation_with(), Parma_Polyhedra_-Library::Grid::relation_with(), Parma_Polyhedra_Library::Box< ITV >::relation_with(), Parma_-Polyhedra_Library::BD_Shape< T >::relation_with(), Parma_Polyhedra_Library::Congruence_-System::satisfies_all_congruences(), Parma_Polyhedra_Library::Grid_Generator::scale_to_divisor(), Parma_Polyhedra_Library::shrink_to_congruence_no_check(), Parma_Polyhedra_Library::Scalar_-Products::sign(), Parma_Polyhedra_Library::Grid::simplify_using_context_assign(), Parma_Polyhedra_-Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::MIP_Problem::solve_mip(), Parma_-Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index(), Parma_Polyhedra_-Library::PIP_Solution_Node::update_solution(), Parma_Polyhedra_Library::Grid::wrap_assign(), and Parma_Polyhedra_Library::Box< ITV >::wrap_assign().

12.88.1.3 #define PPL_OUTPUT_2_PARAM_TEMPLATE_DEFINITIONS(type_symbol1, type_symbol2, class_prefix)

Definition at line 289 of file globals.defs.hh.

12.88.1.4 #define PPL_OUTPUT_3_PARAM_TEMPLATE_DEFINITIONS(type_symbol1, type_symbol2, type_symbol3, class_prefix)

Definition at line 306 of file globals.defs.hh.

12.88.1.5 #define PPL_OUTPUT_DECLARATIONS

Definition at line 241 of file globals.defs.hh.

12.88.1.6 #define PPL_OUTPUT_DEFINITIONS(class_name)

Definition at line 249 of file globals.defs.hh.

12.88.1.7 #define PPL_OUTPUT_DEFINITIONS_ASCII_ONLY(class_name)

Definition at line 261 of file globals.defs.hh.

12.88.1.8 #define PPL_OUTPUT_TEMPLATE_DEFINITIONS(type_symbol, class_prefix)

Definition at line 273 of file globals.defs.hh.

12.88.1.9 #define PPL_OUTPUT_TEMPLATE_DEFINITIONS_ASCII_ONLY(type_symbol, class_prefix)

Definition at line 324 of file globals.defs.hh.

12.88.1.10 #define PPL_PROFILE_ADD_WEIGHT 0

Definition at line 136 of file globals.defs.hh.

12.88.1.11 #define PPL_STR(s) #s

Definition at line 237 of file globals.defs.hh.

12.88.1.12 #define PPL_XSTR(s) PPL_STR(s)

Definition at line 239 of file globals.defs.hh.

12.88.1.13 #define WEIGHT_ADD(delta)

Value:

```
do {                                          \
    if (!Implementation::in_assert)             \
      Weightwatch_Traits::weight += delta;      \
  } while(0)
```

Definition at line 165 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::MIP_Problem::get_exiting_base_index(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

12.88.1.14 #define WEIGHT_ADD_MUL(delta, factor)

Value:

```
do {                                                \
    if (!Implementation::in_assert)                   \
      Weightwatch_Traits::weight += delta * factor;   \
  } while(0)
```

Definition at line 170 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint(), Parma_Polyhedra_-Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::MIP_-Problem::linear_combine(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::normalize(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_no_check(), Parma_Polyhedra_-Library::PIP_Solution_Node::Tableau::scale(), Parma_Polyhedra_Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_entering_index(), and Parma_-Polyhedra_Library::MIP_Problem::steepest_edge_float_entering_index().

12.88.1.15 #define WEIGHT_BEGIN()

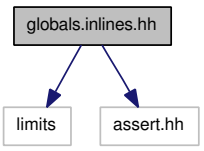
Definition at line 164 of file globals.defs.hh.

Referenced by Parma_Polyhedra_Library::PIP_Tree_Node::add_constraint(), Parma_Polyhedra_-Library::PIP_Tree_Node::compatibility_check(), Parma_Polyhedra_Library::MIP_Problem::compute_-simplex_using_steepest_edge_float(), Parma_Polyhedra_Library::Polyhedron::conversion(), Parma_-Polyhedra_Library::PIP_Solution_Node::generate_cut(), Parma_Polyhedra_Library::MIP_Problem::get_-exiting_base_index(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::is_better_pivot(), Parma_Polyhedra_Library::MIP_Problem::linear_combine(), Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau::normalize(), Parma_Polyhedra_Library::Box< ITV >::propagate_constraints_-no_check(), Parma_Polyhedra_Library::PIP_Solution_Node::Tableau::scale(), Parma_Polyhedra_-Library::PIP_Solution_Node::solve(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_exact_-entering_index(), Parma_Polyhedra_Library::MIP_Problem::steepest_edge_float_entering_index(), and Parma_Polyhedra_Library::PIP_Solution_Node::update_tableau().

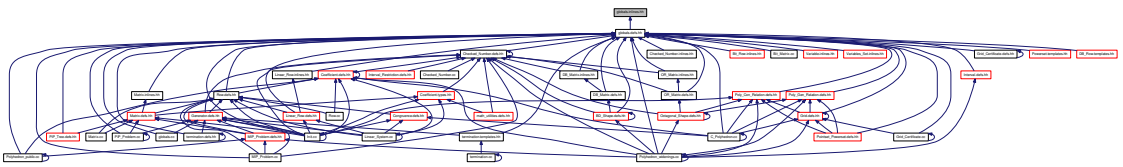
12.89 globals.inlines.hh File Reference

```
#include <limits>
#include "assert.hh"
```

Include dependency graph for globals.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

Functions

- dimension_type Parma_Polyhedra_Library::not_a_dimension ()
  Returns a value that does not designate a valid dimension.
- void Parma_Polyhedra_Library::maybe_abandon ()
- dimension_type Parma_Polyhedra_Library::compute_capacity (dimension_type requested_size, dimension_type maximum_size)
  Speculative allocation function.

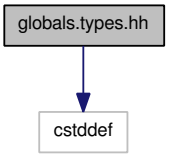
Memory Size Inspection Functions

- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_Library::external_memory_in_bytes (const T &)
  For native types, returns the size in bytes of the memory managed by the type of the (unused) parameter, i.e., 0.
- template<typename T >
  Enable_If< Is_Native< T >::value, memory_size_type >::type Parma_Polyhedra_Library::total_memory_in_bytes (const T &)
  For native types, returns the total size in bytes of the memory occupied by the type of the (unused) parameter, i.e., 0.
- memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpz_class &x)
  Returns the size in bytes of the memory managed by x.
- memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpz_class &x)
  Returns the total size in bytes of the memory occupied by x.
- memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const mpq_class &x)
  Returns the size in bytes of the memory managed by x.
- memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const mpq_class &x)
  Returns the total size in bytes of the memory occupied by x.

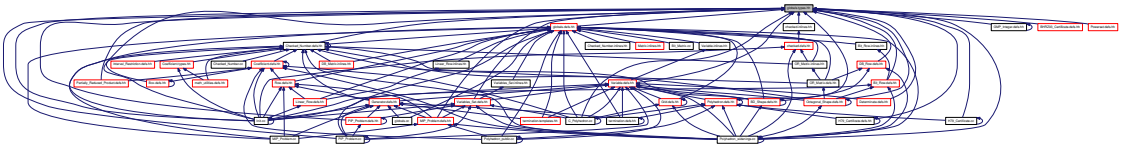
12.90 globals.types.hh File Reference

```
#include <cstddef>
```

Include dependency graph for globals.types.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Typedefs

- typedef size_t Parma_Polyhedra_Library::dimension_type

  An unsigned integral type for representing space dimensions.

- typedef size_t Parma_Polyhedra_Library::memory_size_type

  An unsigned integral type for representing memory size in bytes.

Enumerations

- enum Parma_Polyhedra_Library::Degenerate_Element { Parma_Polyhedra_Library::UNIVERSE, Parma_Polyhedra_Library::EMPTY }

  Kinds of degenerate abstract elements.

- enum Parma_Polyhedra_Library::Relation_Symbol {

  Parma_Polyhedra_Library::EQUAL, Parma_Polyhedra_Library::LESS_THAN, Parma_Polyhedra_Library::LESS_OR_EQUAL, Parma_Polyhedra_Library::GREATER_THAN,

  Parma_Polyhedra_Library::GREATER_OR_EQUAL, Parma_Polyhedra_Library::NOT_EQUAL }

  Relation symbols.

- enum Parma_Polyhedra_Library::Complexity_Class { Parma_Polyhedra_Library::POLYNOMIAL_COMPLEXITY, Parma_Polyhedra_Library::SIMPLEX_COMPLEXITY, Parma_Polyhedra_Library::ANY_COMPLEXITY }

  Complexity pseudo-classes.

- enum Parma_Polyhedra_Library::Optimization_Mode { Parma_Polyhedra_Library::MINIMIZATION, Parma_Polyhedra_Library::MAXIMIZATION }

  Possible optimization modes.

- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Width {

  Parma_Polyhedra_Library::BITS_8, Parma_Polyhedra_Library::BITS_16, Parma_Polyhedra_Library::BITS_32, Parma_Polyhedra_Library::BITS_64,

  Parma_Polyhedra_Library::BITS_128 }

Widths of bounded integer types.

- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Representation { Parma_Polyhedra_Library::UNSIGNED, Parma_Polyhedra_Library::SIGNED_2_COMPLEMENT }

  Representation of bounded integer types.

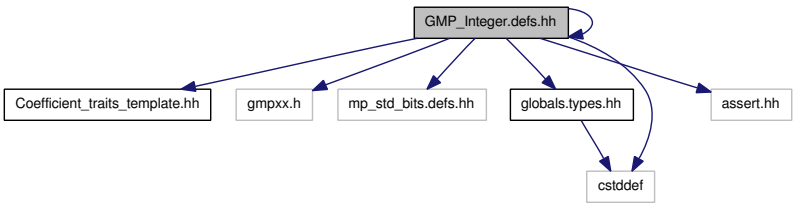
- enum Parma_Polyhedra_Library::Bounded_Integer_Type_Overflow { Parma_Polyhedra_Library::OVERFLOW_WRAPS, Parma_Polyhedra_Library::OVERFLOW_UNDEFINED, Parma_Polyhedra_Library::OVERFLOW_IMPOSSIBLE }

  Overflow behavior of bounded integer types.

12.91 GMP_Integer.defs.hh File Reference

```
#include "GMP_Integer.types.hh"
#include "Coefficient_traits_template.hh"
#include <gmpxx.h>
#include "mp_std_bits.defs.hh"
#include "globals.types.hh"
#include <cstddef>
#include "assert.hh"
```

Include dependency graph for GMP_Integer.defs.hh:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

Memory Size Inspection Functions

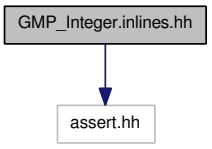
- memory_size_type Parma_Polyhedra_Library::total_memory_in_bytes (const GMP_Integer &x)

- memory_size_type Parma_Polyhedra_Library::external_memory_in_bytes (const GMP_Integer &x)

12.92 GMP_Integer.inlines.hh File Reference

```
#include "assert.hh"
```

Include dependency graph for GMP_Integer.inlines.hh:



Namespaces

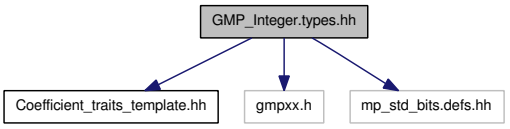
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.93 GMP_Integer.types.hh File Reference

```
#include "Coefficient_traits_template.hh"
#include <gmpxx.h>
#include "mp_std_bits.defs.hh"
```

Include dependency graph for GMP_Integer.types.hh:



Classes

- struct Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer >

  Coefficient traits specialization for unbounded integers.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Typedefs

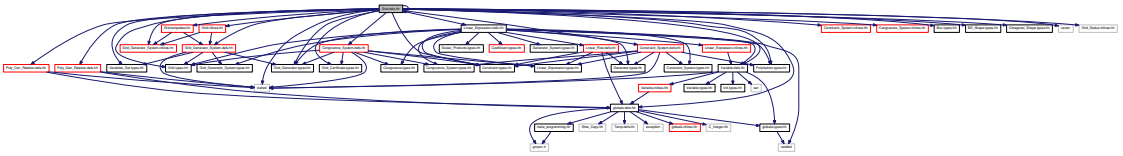
- typedef mpz_class Parma_Polyhedra_Library::GMP_Integer

12.94 gpl.dox File Reference

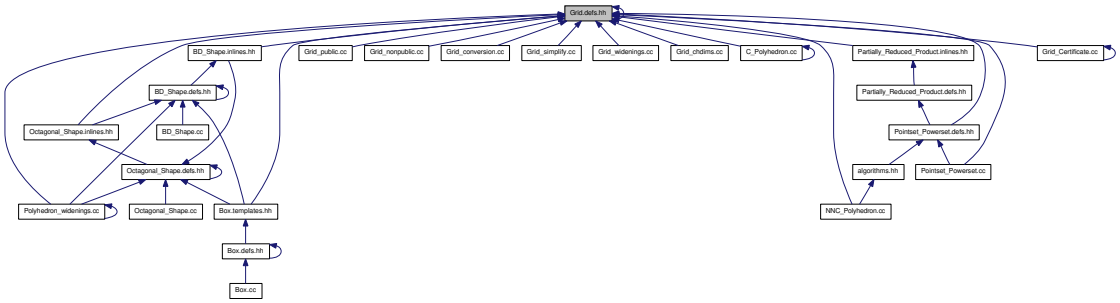
12.95 Grid.defs.hh File Reference

```
#include "Grid.types.hh"
#include "globals.defs.hh"
#include "Variable.defs.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.types.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include "Grid_Generator.types.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "Grid_Certificate.types.hh"
#include "Box.types.hh"
#include "Polyhedron.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include <vector>
#include <iosfwd>
#include "Grid_Status.idefs.hh"
#include "Grid_Status.inlines.hh"
#include "Grid.inlines.hh"
#include "Grid.templates.hh"
```

Include dependency graph for Grid.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Grid

  A grid.

- class Parma_Polyhedra_Library::Grid::Status

  A conjunctive assertion about a grid.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_IN_Grid_CLASS

12.95.1 Define Documentation

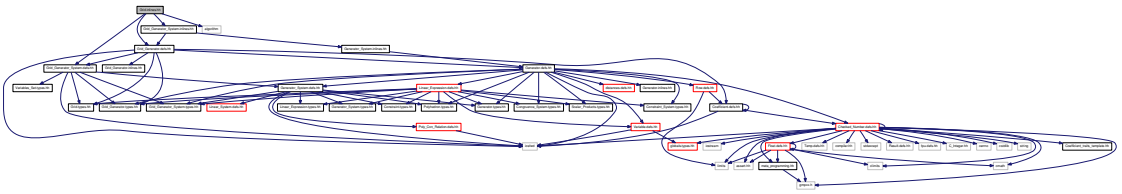
12.95.1.1 #define PPL_IN_Grid_CLASS

Definition at line 1971 of file Grid.defs.hh.

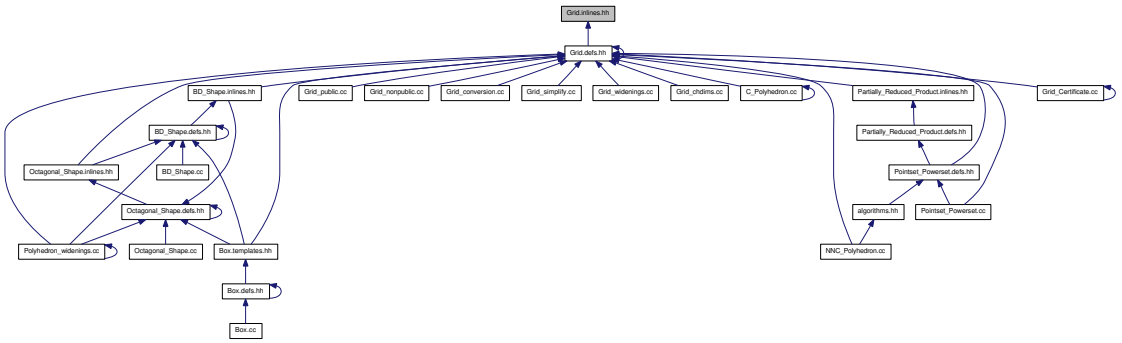
12.96 Grid.inlines.hh File Reference

```
#include "Grid_Generator.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include <algorithm>
```

Include dependency graph for Grid.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

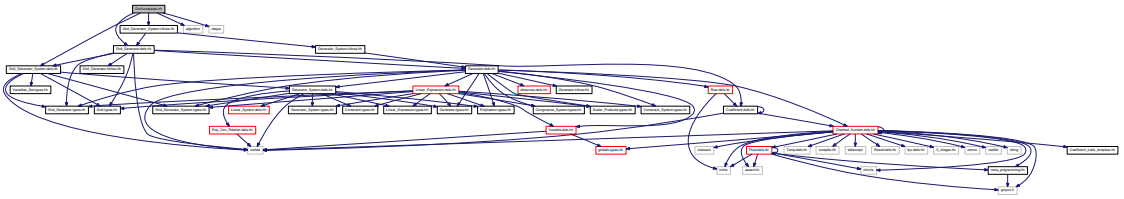
- namespace std

  The standard C++ namespace.

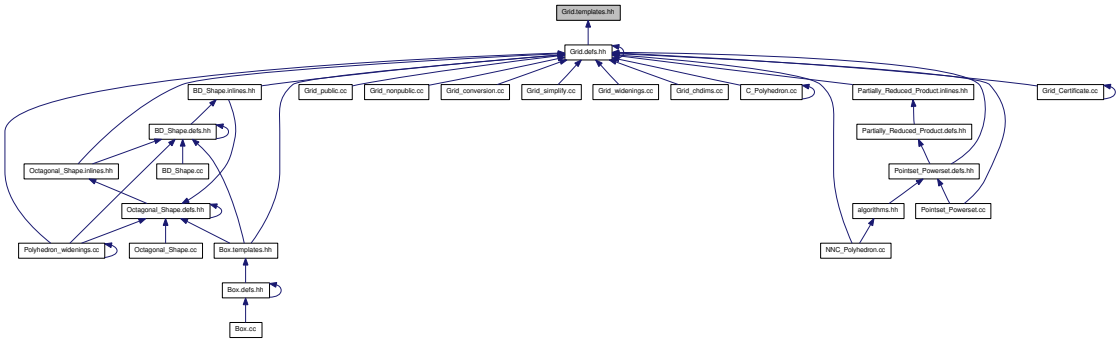
12.97 Grid.templates.hh File Reference

```
#include "Grid_Generator.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include <algorithm>
#include <deque>
```

Include dependency graph for Grid.templates.hh:



This graph shows which files directly or indirectly include this file:



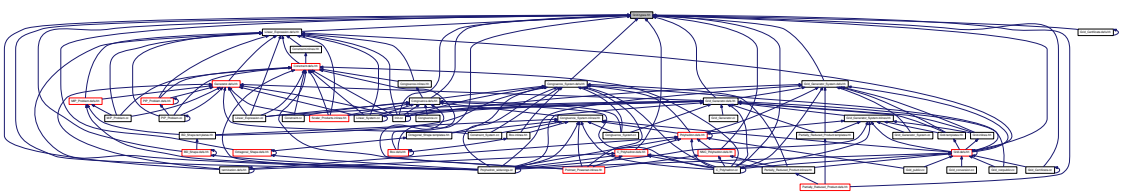
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.98 Grid.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

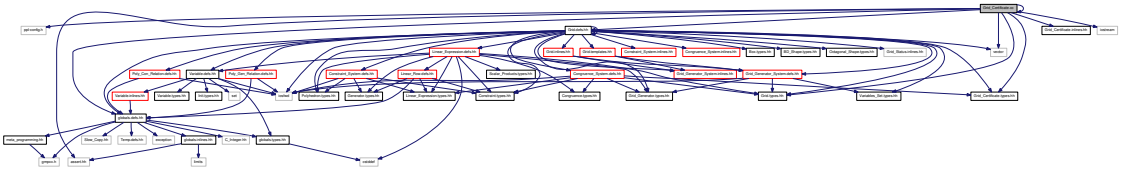
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.99 Grid_Certificate.cc File Reference

```
#include <ppl-config.h>
#include "Grid_Certificate.defs.hh"
#include "Grid_Certificate.types.hh"
#include "Grid.types.hh"
#include "globals.defs.hh"
#include "assert.hh"
#include <vector>
#include "Grid_Certificate.inlines.hh"
#include "Grid.defs.hh"
#include <iostream>
```

Include dependency graph for Grid_Certificate.cc:

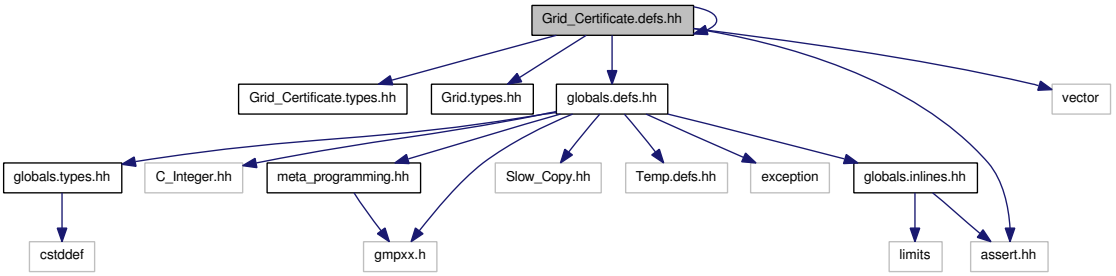


This graph shows which files directly or indirectly include this file:

12.100 Grid_Certificate.defs.hh File Reference

```
#include "Grid_Certificate.types.hh"
#include "Grid.types.hh"
#include "globals.defs.hh"
#include "assert.hh"
#include <vector>
#include "Grid_Certificate.inlines.hh"
```

Include dependency graph for Grid_Certificate.defs.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class Parma_Polyhedra_Library::Grid_Certificate

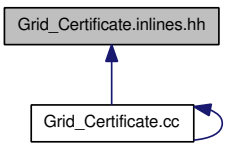
  The convergence certificate for the Grid widening operator.

- struct Parma_Polyhedra_Library::Grid_Certificate::Compare

  A total ordering on Grid certificates.

12.101 Grid_Certificate.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



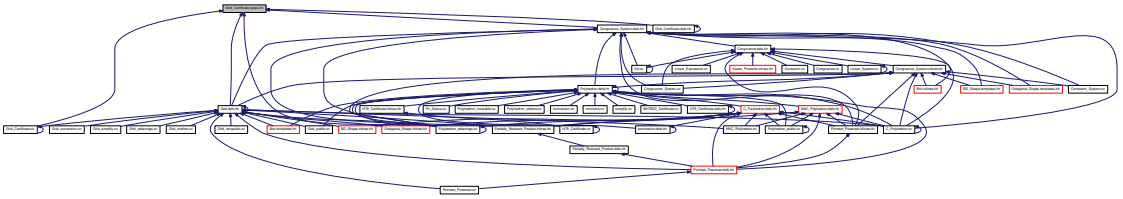
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.102 Grid_Certificate.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

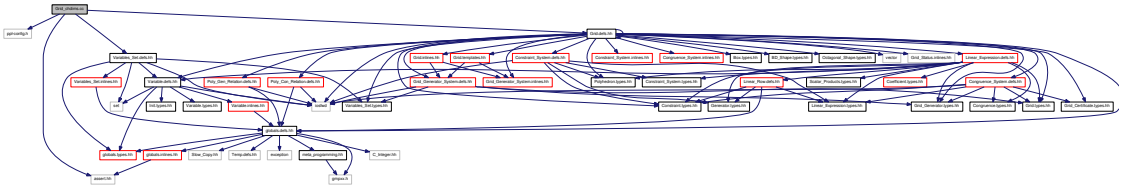
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.103 Grid_chdims.cc File Reference

```
#include <ppl-config.h>
#include "Grid.defs.hh"
#include "Variables_Set.defs.hh"
#include "assert.hh"
```

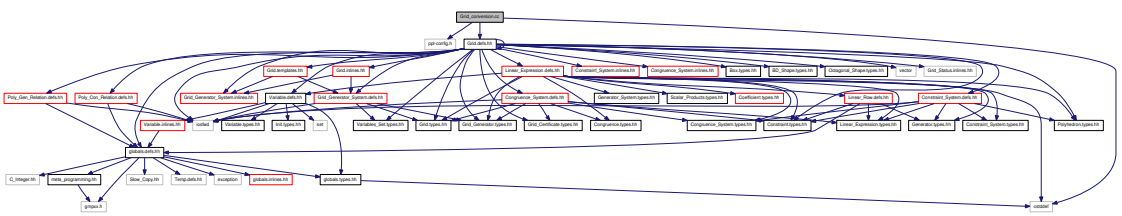
Include dependency graph for Grid_chdims.cc:



12.104 Grid_conversion.cc File Reference

```
#include <ppl-config.h>
#include "Grid.defs.hh"
#include <cstddef>
```

Include dependency graph for Grid_conversion.cc:



Namespaces

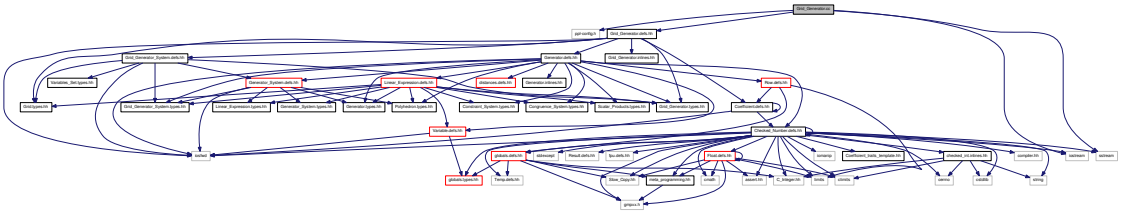
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.105 Grid_Generator.cc File Reference

```
#include <ppl-config.h>
#include "Grid_Generator.defs.hh"
#include <iostream>
#include <sstream>
```

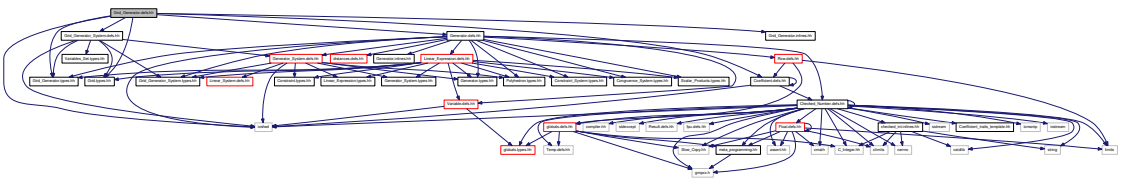
Include dependency graph for Grid_Generator.cc:



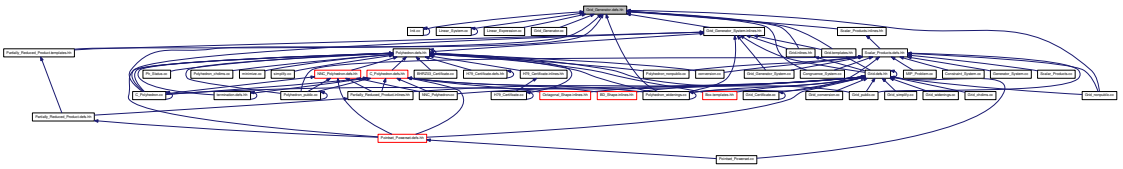
12.106 Grid_Generator.defs.hh File Reference

```
#include "Grid_Generator.types.hh"
#include "Coefficient.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Generator.defs.hh"
#include "Grid.types.hh"
#include <iosfwd>
#include "Grid_Generator.inlines.hh"
```

Include dependency graph for Grid_Generator.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Grid_Generator

  A grid line, parameter or grid point.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

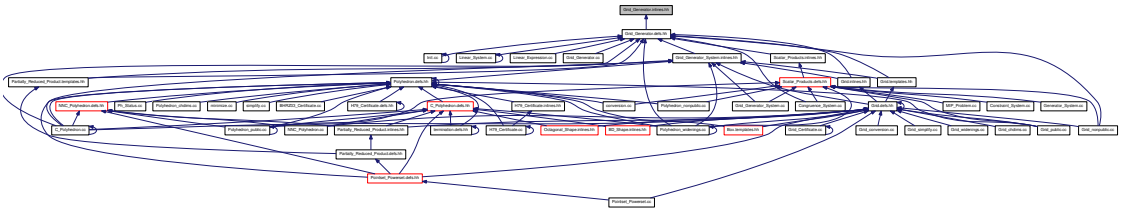
  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

12.107 Grid_Generator.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

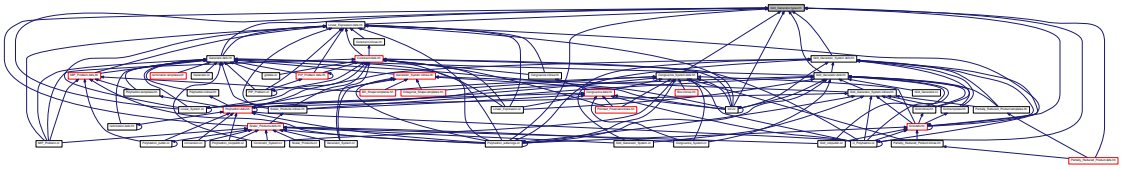
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.108 Grid_Generator.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

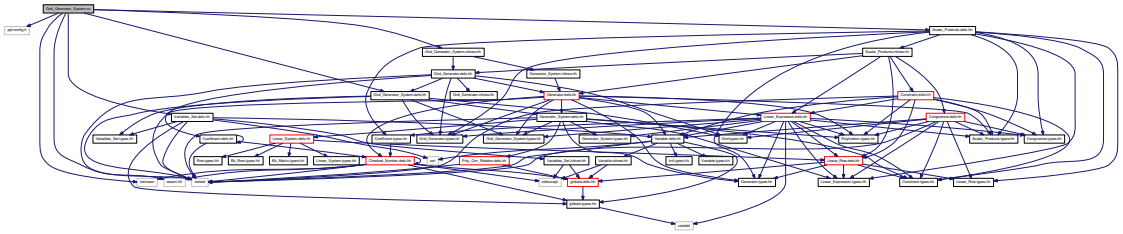
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.109 Grid_Generator_System.cc File Reference

```
#include <ppl-config.h>
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include "Scalar_Products.defs.hh"
#include "Variables_Set.defs.hh"
#include "assert.hh"
#include <iostream>
```

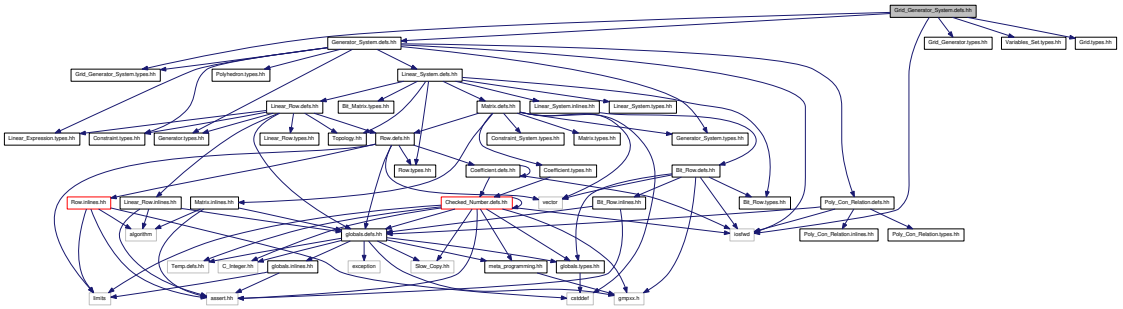
Include dependency graph for Grid_Generator_System.cc:



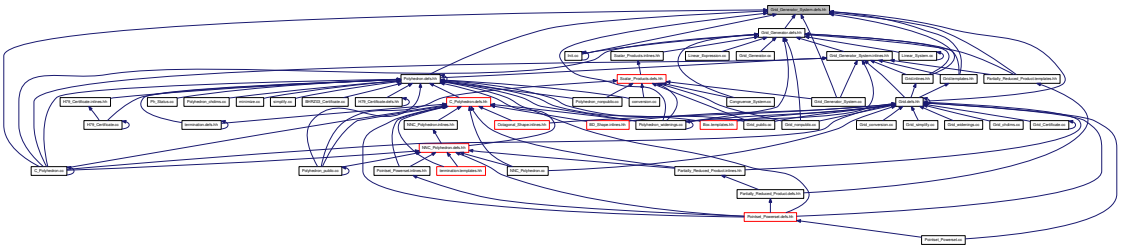
12.110 Grid_Generator_System.defs.hh File Reference

```
#include "Grid_Generator_System.types.hh"
#include "Generator_System.defs.hh"
#include "Grid_Generator.types.hh"
#include "Variables_Set.types.hh"
#include "Grid.types.hh"
#include <iosfwd>
```

Include dependency graph for Grid_Generator_System.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Grid_Generator_System
  A system of grid generators.
- class Parma_Polyhedra_Library::Grid_Generator_System::const_iterator
  An iterator over a system of grid generators.

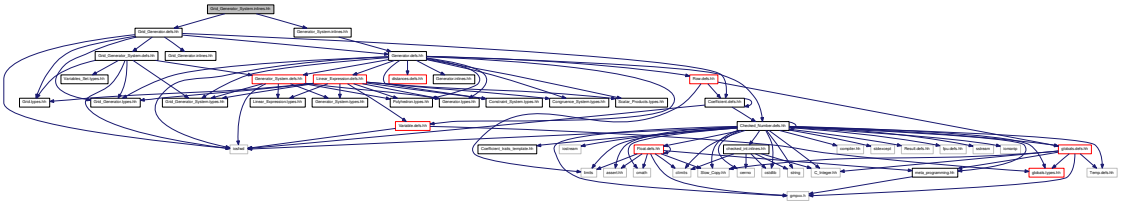
Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.
- namespace Parma_Polyhedra_Library::IO_Operators
  All input/output operators are confined to this namespace.
- namespace std
  The standard C++ namespace.

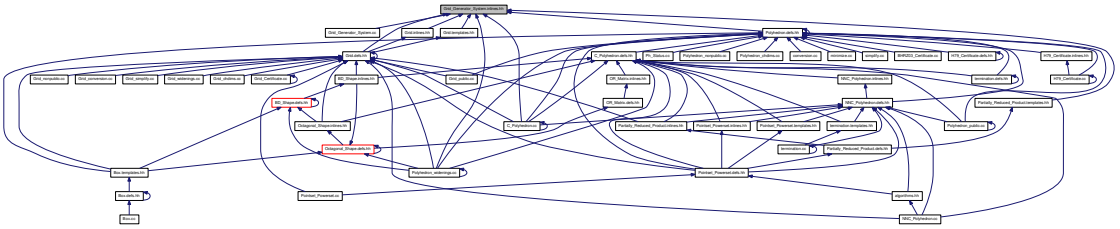
12.111 Grid_Generator_System.inlines.hh File Reference

```
#include "Grid_Generator.defs.hh"
#include "Generator_System.inlines.hh"
```

Include dependency graph for Grid_Generator_System.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

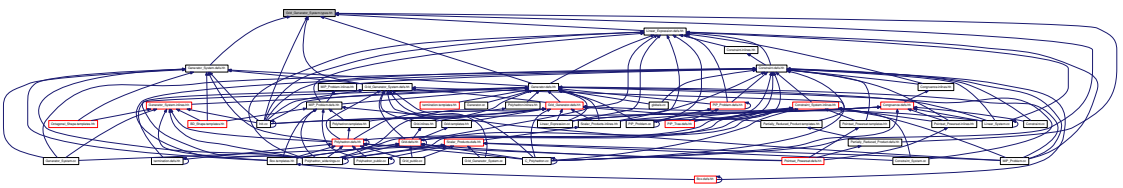
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.112 Grid_Generator_System.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

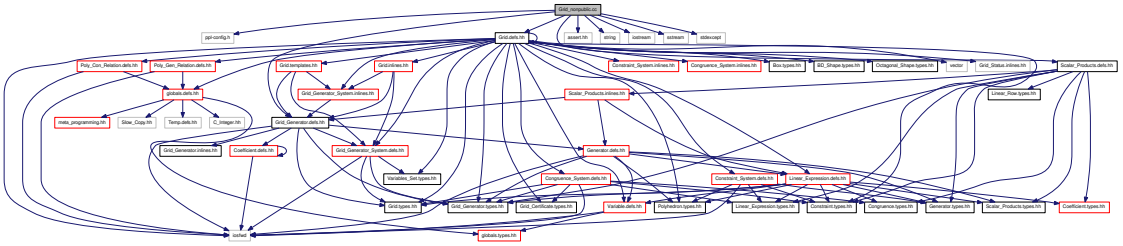
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.113 Grid_nonpublic.cc File Reference

```
#include <ppl-config.h>
#include "Grid.defs.hh"
#include "Grid_Generator.defs.hh"
#include "Scalar_Products.defs.hh"
#include "assert.hh"
#include <string>
#include <iostream>
#include <sstream>
#include <stdexcept>
```

Include dependency graph for Grid_nonpublic.cc:



Defines

- #define BE_LAZY 1

  Controls the laziness level of the implementation.

12.113.1 Define Documentation

12.113.1.1 #define BE_LAZY 1

Controls the laziness level of the implementation.

Temporarily used in a few of the function implementations to switch to an even more lazy algorithm. To be removed as soon as we collect enough information to decide which is the better implementation alternative.

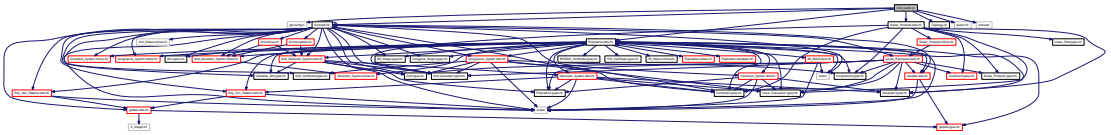
Definition at line 47 of file Grid_nonpublic.cc.

12.114 Grid_public.cc File Reference

```
#include <ppl-config.h>
#include "Grid.defs.hh"
#include "Topology.hh"
#include "Scalar_Products.defs.hh"
```

```
#include "Polyhedron.defs.hh"
#include "assert.hh"
#include <iostream>
```

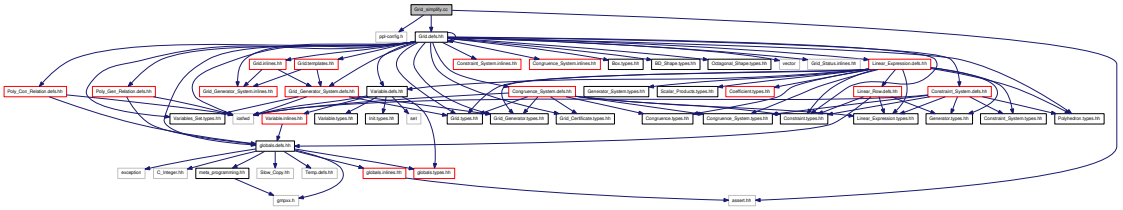
Include dependency graph for Grid_public.cc:



12.115 Grid_simplify.cc File Reference

```
#include <ppl-config.h>
#include "assert.hh"
#include "Grid.defs.hh"
```

Include dependency graph for Grid_simplify.cc:



Namespaces

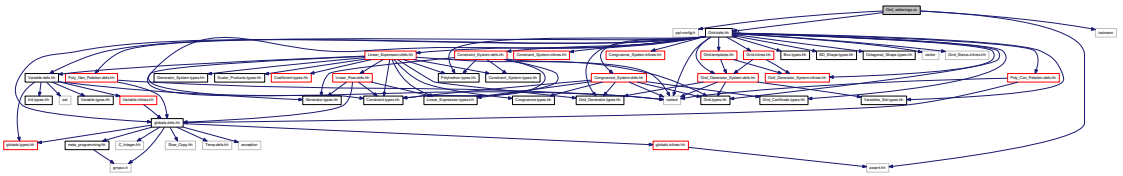
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.116 Grid_widenings.cc File Reference

```
#include <ppl-config.h>
#include "Grid.defs.hh"
#include "assert.hh"
#include <iostream>
```

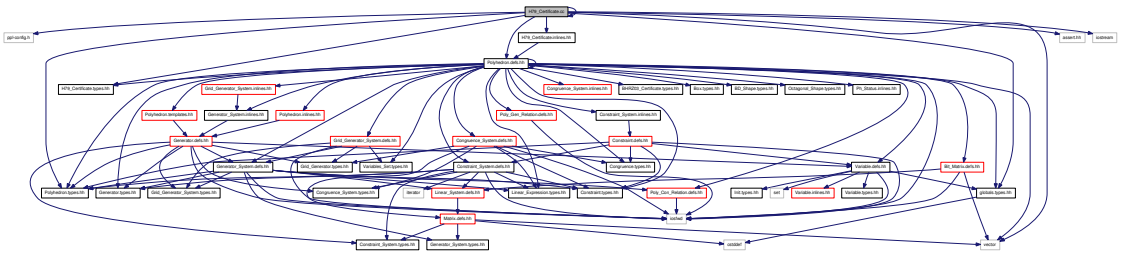
Include dependency graph for Grid_widenings.cc:



12.117 H79_Certificate.cc File Reference

```
#include <ppl-config.h>
#include "H79_Certificate.defs.hh"
#include "H79_Certificate.types.hh"
#include "Polyhedron.types.hh"
#include "globals.types.hh"
#include "assert.hh"
#include <vector>
#include "H79_Certificate.inlines.hh"
#include "Polyhedron.defs.hh"
#include <iostream>
```

Include dependency graph for H79_Certificate.cc:

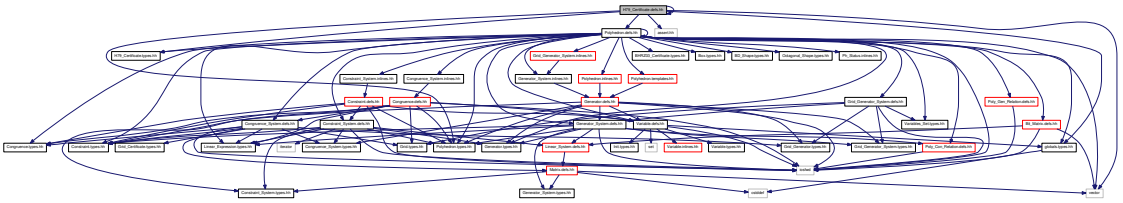


This graph shows which files directly or indirectly include this file:

12.118 H79_Certificate.defs.hh File Reference

```
#include "H79_Certificate.types.hh"
#include "Polyhedron.types.hh"
#include "globals.types.hh"
#include "assert.hh"
#include <vector>
#include "H79_Certificate.inlines.hh"
#include "Polyhedron.defs.hh"
```

Include dependency graph for H79_Certificate.defs.hh:



This graph shows which files directly or indirectly include this file:

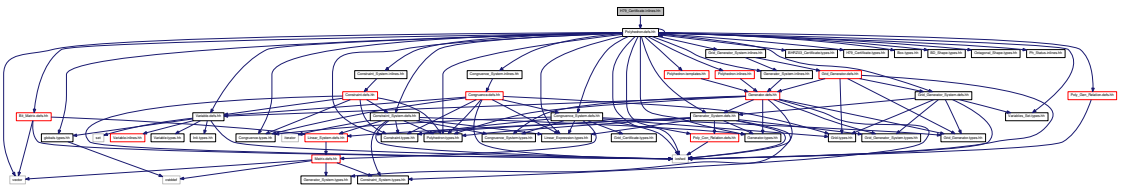
Classes

- class Parma_Polyhedra_Library::H79_Certificate
  A convergence certificate for the H79 widening operator.
- struct Parma_Polyhedra_Library::H79_Certificate::Compare
  A total ordering on H79 certificates.

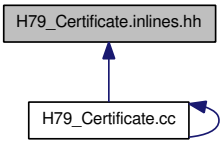
12.119 H79_Certificate.inlines.hh File Reference

```
#include "Polyhedron.defs.hh"
```

Include dependency graph for H79_Certificate.inlines.hh:



This graph shows which files directly or indirectly include this file:

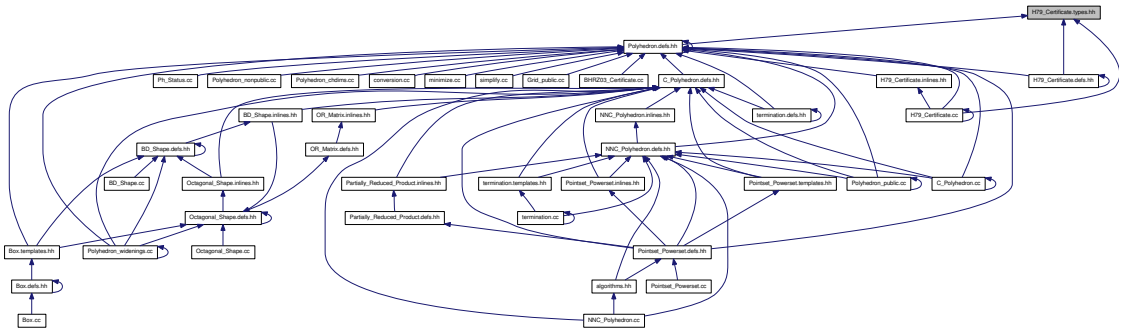


Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

12.120 H79_Certificate.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.121 Init.cc File Reference

```
#include <ppl-config.h>
#include "Init.defs.hh"
#include "Init.types.hh"
#include "fpu.types.hh"
#include "Init.inlines.hh"
#include "globals.types.hh"
#include <iosfwd>
#include <set>
#include "globals.defs.hh"
#include <stdexcept>
#include "fpu.defs.hh"
#include "Rounding_Dir.defs.hh"
#include "checked.defs.hh"
#include "Coefficient.defs.hh"
#include <vector>
#include <limits>
#include "Coefficient.types.hh"
#include "Checked_Number.defs.hh"
#include <gmpxx.h>
```

```
#include "assert.hh"
#include <cstddef>
#include <algorithm>
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Variable.defs.hh"
#include "Scalar_Products.types.hh"
#include "Linear_Row.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint_System.types.hh"
#include "Row.types.hh"
#include "Row.defs.hh"
#include "Generator_System.types.hh"
#include "Topology.hh"
#include "Bit_Row.types.hh"
#include "Polyhedron.types.hh"
#include "Congruence_System.types.hh"
#include <iterator>
#include "Congruence.types.hh"
#include "Grid_Generator_System.types.hh"
#include "Linear_System.defs.hh"
#include "Grid_Generator.types.hh"
#include "Result.defs.hh"
#include "Grid.types.hh"
#include "Linear_Row.types.hh"
#include "Matrix.defs.hh"
#include "Constraint.defs.hh"
#include <sstream>
#include "Generator_System.defs.hh"
#include "Generator.defs.hh"
#include "Constraint_System.defs.hh"
#include "Congruence_System.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Variables_Set.types.hh"
#include "Congruence.defs.hh"
```

```
#include "Grid_Generator.defs.hh"
#include "Generator_System.inlines.hh"
#include "Bit_Matrix.types.hh"
#include "Bit_Row.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "compiler.hh"
#include <deque>
#include "Variables_Set.defs.hh"
#include "meta_programming.hh"
#include "Slow_Copy.hh"
#include <cstdlib>
#include "assign_or_swap.hh"
#include <iostream>
#include <iomanip>
```

This graph shows which files directly or indirectly include this file:

Functions

- void ppl_set_GMP_memory_allocation_functions (void)

12.121.1 Function Documentation

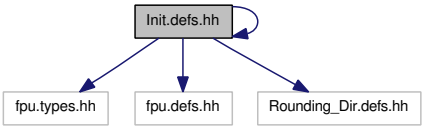
12.121.1.1 void ppl_set_GMP_memory_allocation_functions (void)

Referenced by Parma_Polyhedra_Library::Init::Init().

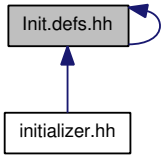
12.122 Init.defs.hh File Reference

```
#include "Init.types.hh"
#include "fpu.types.hh"
#include "fpu.defs.hh"
#include "Rounding_Dir.defs.hh"
```

Include dependency graph for Init.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Init
  Class for initialization and finalization.

Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

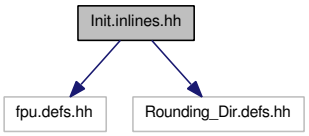
Functions

- void Parma_Polyhedra_Library::set_rounding_for_PPL ()
  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.
- void Parma_Polyhedra_Library::restore_pre_PPL_rounding ()
  Sets the FPU rounding mode as it was before initialization of the PPL.

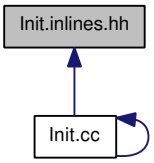
12.123 Init.inlines.hh File Reference

```
#include "fpu.defs.hh"
#include "Rounding_Dir.defs.hh"
```

Include dependency graph for Init.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- void Parma_Polyhedra_Library::set_rounding_for_PPL ()

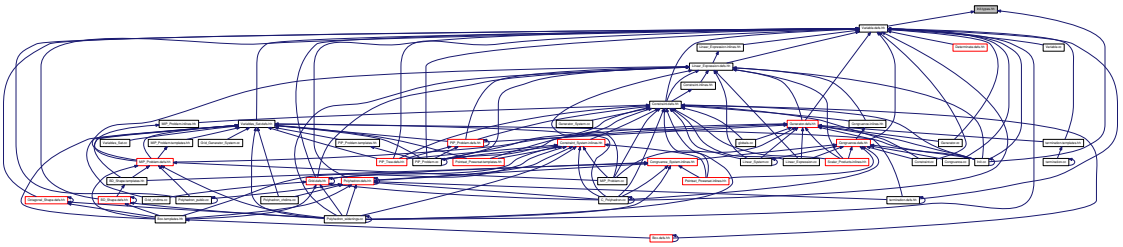
  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

- void Parma_Polyhedra_Library::restore_pre_PPL_rounding ()

  Sets the FPU rounding mode as it was before initialization of the PPL.

12.124 Init.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

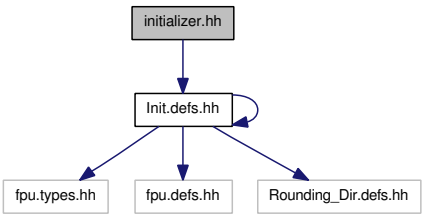
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.125 initializer.hh File Reference

```
#include "Init.defs.hh"
```

Include dependency graph for initializer.hh:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- void Parma_Polyhedra_Library::initialize ()

  Initializes the library.

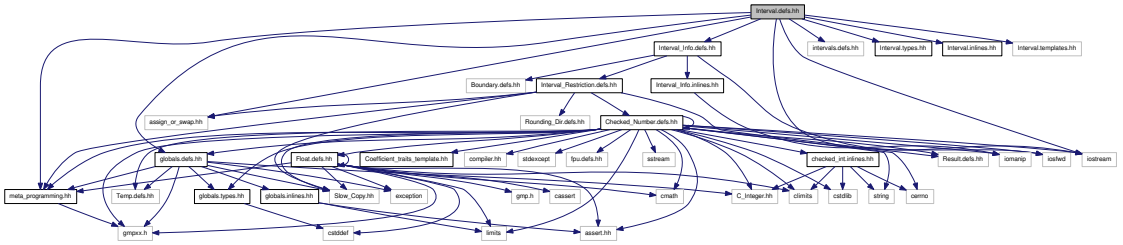
- void Parma_Polyhedra_Library::finalize ()

  Finalizes the library.

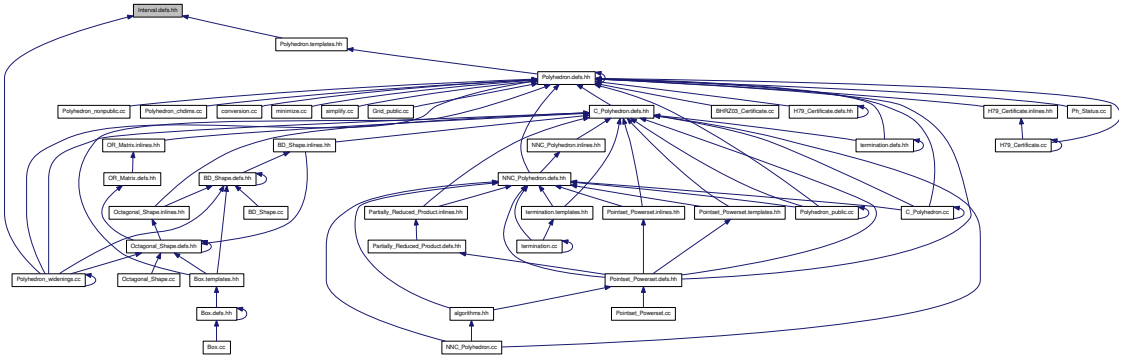
12.126 Interval.defs.hh File Reference

```
#include "globals.defs.hh"
#include "meta_programming.hh"
#include "assign_or_swap.hh"
#include "intervals.defs.hh"
#include "Interval.types.hh"
#include "Interval_Info.defs.hh"
#include <iosfwd>
#include <iostream>
#include "Interval.inlines.hh"
#include "Interval.templates.hh"
```

Include dependency graph for Interval.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Interval_Base
- struct Parma_Polyhedra_Library::Is_Singleton< T, Enable >
- struct Parma_Polyhedra_Library::Is_Interval< T >
- class Parma_Polyhedra_Library::Interval< Boundary, Info >

  A generic, not necessarily closed, possibly restricted interval.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Enumerations

- enum Parma_Polyhedra_Library::Ternary { Parma_Polyhedra_Library::T_YES, Parma_Polyhedra_Library::T_NO, Parma_Polyhedra_Library::T_MAYBE }

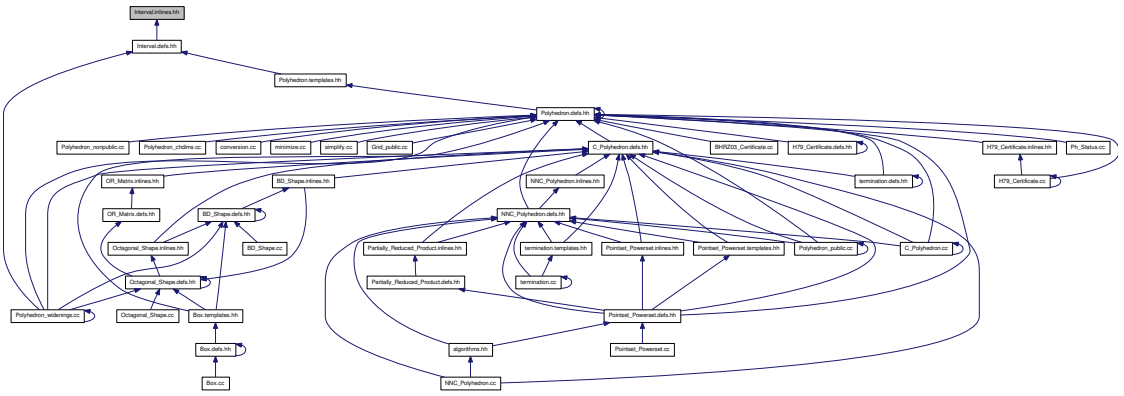
Functions

- I_Result Parma_Polyhedra_Library::combine (Result l, Result u)

- template<typename From >
  Enable_If< Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::neg_assign (From &x)

12.127 Interval.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< Interval_Boundary_Type >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< float >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< char >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed char >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned char >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed short >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned short >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed int >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned int >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< signed long >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long >
- struct Parma_Polyhedra_Library::Select_Temp_Boundary_Type< unsigned long long >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Interval_NS
- namespace std

  The standard C++ namespace.

Typedefs

- typedef Interval_Restriction_None< Interval_Info_Null< Scalar_As_Interval_Policy > > Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Info
- typedef Interval_Restriction_None< Interval_Info_Null_Open< Scalar_As_Interval_Policy > > Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Info_Open

Functions

- template<typename From >
  Enable_If< Is_Interval< From >::value, I_Result >::type Parma_Polyhedra_Library::neg_assign (From &x)
- template<typename Boundary , typename Info >
  bool Parma_Polyhedra_Library::f_is_empty (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  bool Parma_Polyhedra_Library::f_is_singleton (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  int Parma_Polyhedra_Library::is_infinity (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  const Boundary & Parma_Polyhedra_Library::Interval_NS::f_lower (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  const Boundary & Parma_Polyhedra_Library::Interval_NS::f_upper (const Interval< Boundary, Info > &x)
- template<typename Boundary , typename Info >
  const Info & Parma_Polyhedra_Library::Interval_NS::f_info (const Interval< Boundary, Info > &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const T & >::type Parma_Polyhedra_Library::Interval_NS::f_lower (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const T & >::type Parma_Polyhedra_Library::Interval_NS::f_upper (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, const Scalar_As_Interval_Info & >::type Parma_Polyhedra_Library::Interval_NS::f_info (const T &)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, Scalar_As_Interval_Info_Open >::type Parma_Polyhedra_Library::Interval_NS::f_info (const T &, bool open)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, bool >::type Parma_Polyhedra_Library::Interval_NS::f_is_empty (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value, bool >::type Parma_Polyhedra_Library::Interval_NS::f_is_singleton (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type Parma_Polyhedra_Library::is_singleton_integer (const T &x)
- template<typename T >
  Enable_If< Is_Singleton< T >::value||Is_Interval< T >::value, bool >::type Parma_Polyhedra_Library::check_empty_arg (const T &x)

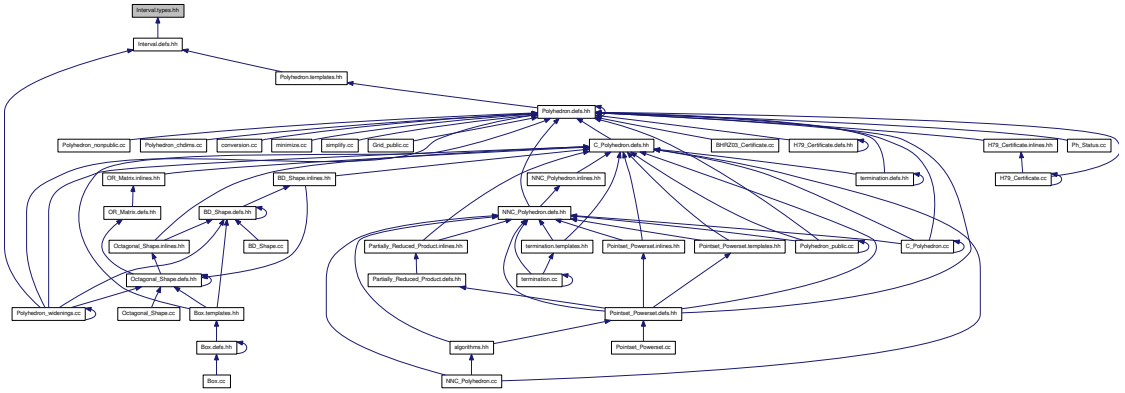
- template<typename T1 , typename T2 >
  Enable_If<((Is_Singleton< T1 >::value||Is_Interval< T1 >::value)&&(Is_Singleton< T2 >::value||Is_Interval< T2 >::value)&&(Is_Interval< T1 >::value||Is_Interval< T2 >::value)), bool >::type Parma_Polyhedra_Library::operator== (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If<((Is_Singleton< T1 >::value||Is_Interval< T1 >::value)&&(Is_Singleton< T2 >::value||Is_Interval< T2 >::value)&&(Is_Interval< T1 >::value||Is_Interval< T2 >::value)), bool >::type Parma_Polyhedra_Library::operator!= (const T1 &x, const T2 &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator+ (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator+ (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > Parma_Polyhedra_Library::operator+ (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator- (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator- (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > Parma_Polyhedra_Library::operator- (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator* (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator* (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > Parma_Polyhedra_Library::operator* (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator/ (const Interval< B, Info > &x, const T &y)
- template<typename B , typename Info , typename T >
  Enable_If< Is_Singleton< T >::value, Interval< B, Info > >::type Parma_Polyhedra_Library::operator/ (const T &x, const Interval< B, Info > &y)
- template<typename B , typename Info >
  Interval< B, Info > Parma_Polyhedra_Library::operator/ (const Interval< B, Info > &x, const Interval< B, Info > &y)
- template<typename Boundary , typename Info >
  std::ostream & Parma_Polyhedra_Library::operator<< (std::ostream &os, const Interval< Boundary, Info > &x)

Variables

- const Scalar_As_Interval_Info Parma_Polyhedra_Library::Interval_NS::SCALAR_INFO

12.128 Interval.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

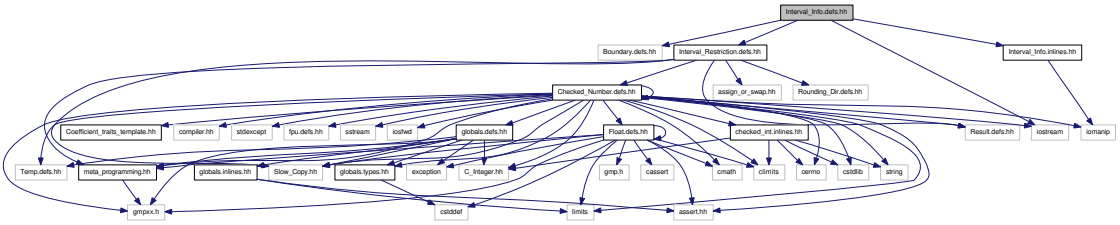
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

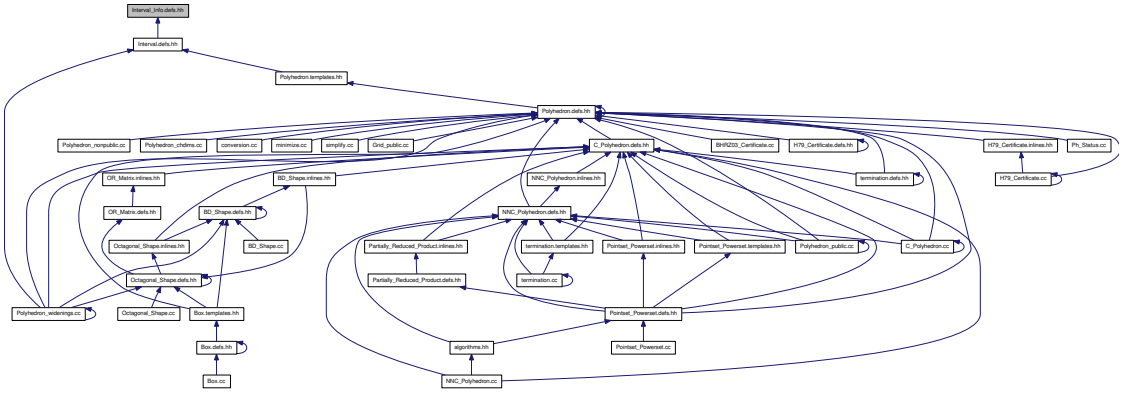
12.129 Interval_Info.defs.hh File Reference

```
#include "Boundary.defs.hh"
#include "Interval_Restriction.defs.hh"
#include <iostream>
#include "Interval_Info.inlines.hh"
```

Include dependency graph for Interval_Info.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Interval_NS::Property
- class Parma_Polyhedra_Library::Interval_Info_Null< Policy >
- class Parma_Polyhedra_Library::Interval_Info_Null_Open< Policy >
- class Parma_Polyhedra_Library::Interval_Info_Bitset< T, Policy >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Interval_NS

Functions

- template<typename T >
  void Parma_Polyhedra_Library::Interval_NS::reset_bits (T &bits)
- template<typename T >
  void Parma_Polyhedra_Library::Interval_NS::reset_bit (T &bits, unsigned int bit)
- template<typename T >
  void Parma_Polyhedra_Library::Interval_NS::set_bit (T &bits, unsigned int bit, bool value)
- template<typename T >
  bool Parma_Polyhedra_Library::Interval_NS::get_bit (const T &bits, unsigned int bit)
- template<typename T >
  void Parma_Polyhedra_Library::Interval_NS::set_bits (T &bits, unsigned int start, unsigned int len, T value)
- template<typename T >
  T Parma_Polyhedra_Library::Interval_NS::get_bits (T &bits, unsigned int start, unsigned int len)

Variables

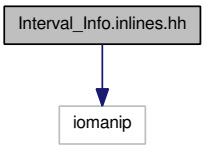
- const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_0 (Property::CARDINALITY_0_)

- const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_1 (Property::CARDINALITY_1_)
- const Property Parma_Polyhedra_Library::Interval_NS::CARDINALITY_IS (Property::CARDINALITY_IS_)

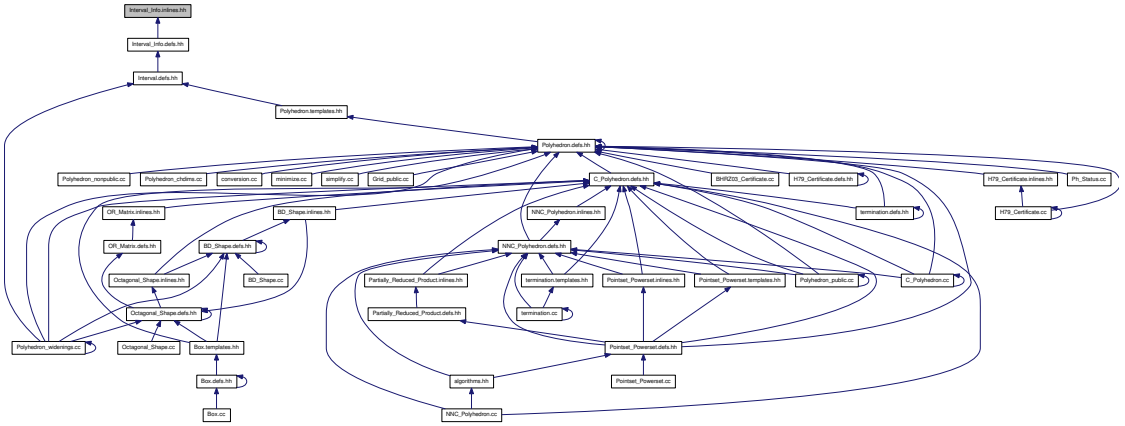
12.130 Interval_Info.inlines.hh File Reference

```
#include <iomanip>
```

Include dependency graph for Interval_Info.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.131 Interval_Info.types.hh File Reference

Namespaces

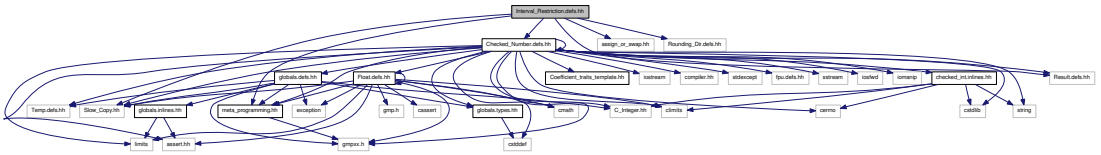
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

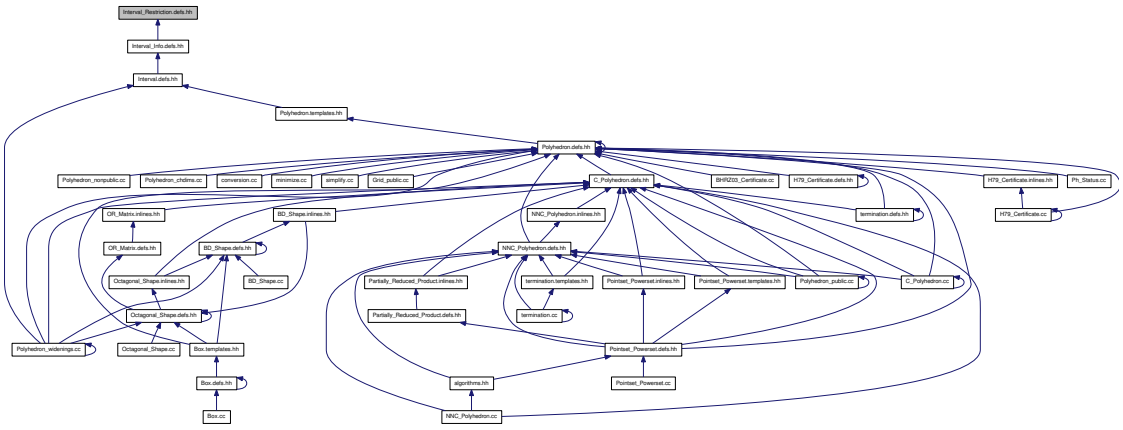
12.132 Interval_Restriction.defs.hh File Reference

```
#include "meta_programming.hh"
#include "Slow_Copy.hh"
#include "assign_or_swap.hh"
#include "Result.defs.hh"
#include "Rounding_Dir.defs.hh"
#include "Checked_Number.defs.hh"
```

Include dependency graph for Interval_Restriction.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Boundary_Value< T, Enable >
- struct Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_Derived< Interval_Base, T >::value >::type >
- class Parma_Polyhedra_Library::Interval_Restriction_None_Base
- class Parma_Polyhedra_Library::Interval_Restriction_None< Base >
- class Parma_Polyhedra_Library::Interval_Restriction_Integer_Base
- class Parma_Polyhedra_Library::Interval_Restriction_Integer< Base >
- class Parma_Polyhedra_Library::Simple_Restriction_Integer
- struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >
- struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >

- struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >
- class Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base
- class Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo< T, Base >
- struct Parma_Polyhedra_Library::Slow_Copy< Interval_Restriction_Integer_Modulo< T, Base > >
- struct Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >
- struct Parma_Polyhedra_Library::Simple_Restriction_Integer_Modulo< T >
- struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >
- struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >
- struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >
- struct Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- bool Parma_Polyhedra_Library::eq_restriction (const Interval_Restriction_None_Base &, const Interval_Restriction_None_Base)
- template<typename T >
  bool Parma_Polyhedra_Library::contains_restriction (const Interval_Restriction_None_Base &, const T &)
- template<typename T >
  bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_None_Base &, const T &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T >
  bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_None_Base &, const T &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)

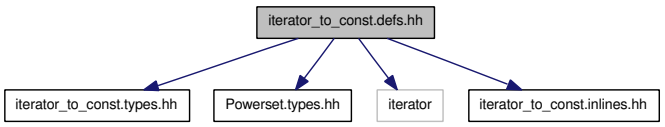
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- template<typename T1 , typename T2 >
  bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_None_Base &, const T1 &, const T2 &)
- void Parma_Polyhedra_Library::output_restriction (std::ostream &, const Interval_Restriction_None_Base &)
- template<typename T1 , typename T2 >
  Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T1 >::value &&Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T2 >::value, bool >::type Parma_Polyhedra_Library::eq_restriction (const T1 &x, const T2 &y)
- template<typename T1 , typename T2 >
  Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T1 >::value &&Is_Same_Or_Derived< Interval_Restriction_Integer_Base, T2 >::value, bool >::type Parma_Polyhedra_Library::contains_restriction (const T1 &x, const T2 &y)
- template<typename Base , typename From >
  bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_Integer< Base > &to, const From &x)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &)
- template<typename Base , typename From >
  bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_Integer< Base > &to, const From &x)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_Integer< Base > &to, const From1 &x, const From2 &y)
- template<typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_Integer< Base > &to, const From1 &, const From2 &)
- template<typename Base >
  void Parma_Polyhedra_Library::output_restriction (std::ostream &s, const Interval_Restriction_Integer< Base > &x)
- template<typename T , typename Base >
  bool Parma_Polyhedra_Library::set_unrestricted (Interval_Restriction_Integer_Modulo< T, Base > &to)

- template<typename T , typename Base >
  bool Parma_Polyhedra_Library::set_integer (Interval_Restriction_Integer_Modulo< T, Base > &to)
- template<typename T , typename Base , typename From >
  bool Parma_Polyhedra_Library::assign_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From &x)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::join_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::intersect_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::diff_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From >
  bool Parma_Polyhedra_Library::neg_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From &x)
- template<typename T >
  void Parma_Polyhedra_Library::addmod (T &to, const T &x, const T &y, const T &to_m, const T &y_m)
- template<typename M , typename T >
  bool Parma_Polyhedra_Library::assign_rem (M &rem, const T &n, const M &div)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::add_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T >
  void Parma_Polyhedra_Library::submod (T &to, const T &x, const T &y, const T &to_m, const T &y_m)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::sub_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T >
  void Parma_Polyhedra_Library::mulmod (T &to, const T &x, const T &y, const T &to_m)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::mul_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base , typename From1 , typename From2 >
  bool Parma_Polyhedra_Library::div_restriction (Interval_Restriction_Integer_Modulo< T, Base > &to, const From1 &x, const From2 &y)
- template<typename T , typename Base >
  void Parma_Polyhedra_Library::output_restriction (std::ostream &s, const Interval_Restriction_Integer_Modulo< T, Base > &x)

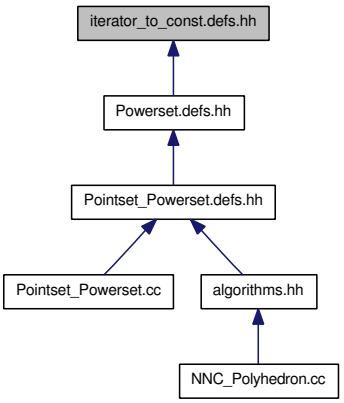
12.133 iterator_to_const.defs.hh File Reference

```
#include "iterator_to_const.types.hh"
#include "Powerset.types.hh"
#include <iterator>
#include "iterator_to_const.inlines.hh"
```

Include dependency graph for iterator_to_const.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::iterator_to_const< Container >

  An iterator on a sequence of read-only objects.

- class Parma_Polyhedra_Library::const_iterator_to_const< Container >

  A const_iterator on a sequence of read-only objects.

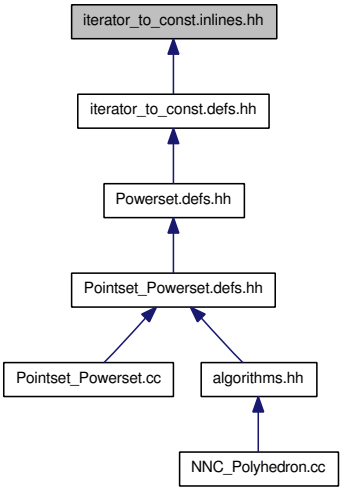
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.134 iterator_to_const.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



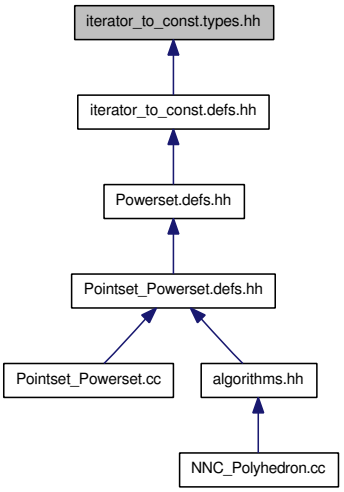
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.135 iterator_to_const.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

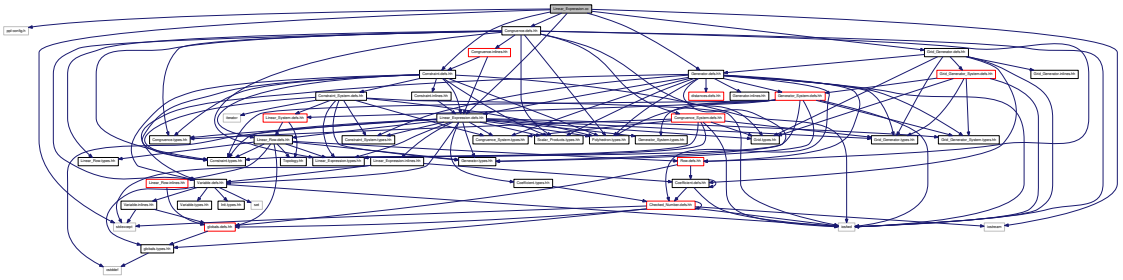
- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

12.136 Linear_Expression.cc File Reference

```
#include <ppl-config.h>
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include "Generator.defs.hh"
#include "Grid_Generator.defs.hh"
#include "Congruence.defs.hh"
#include <stdexcept>
#include <iostream>
```

Include dependency graph for Linear_Expression.cc:

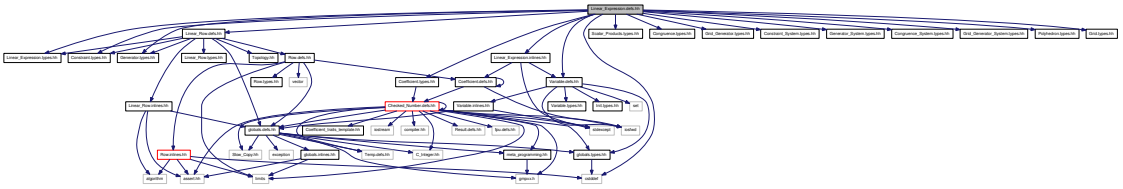


12.137 Linear_Expression.defs.hh File Reference

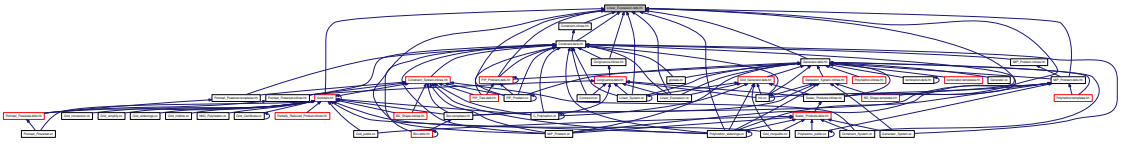
```
#include "Linear_Expression.types.hh"
#include "Scalar_Products.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Grid_Generator.types.hh"
#include "Linear_Row.defs.hh"
#include "Coefficient.types.hh"
#include "Variable.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.types.hh"
#include "Congruence_System.types.hh"
#include "Grid_Generator_System.types.hh"
#include "Polyhedron.types.hh"
```

```
#include "Grid.types.hh"
#include <cstddef>
#include "Linear_Expression.inlines.hh"
```

Include dependency graph for Linear_Expression.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Linear_Expression

  A linear expression.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

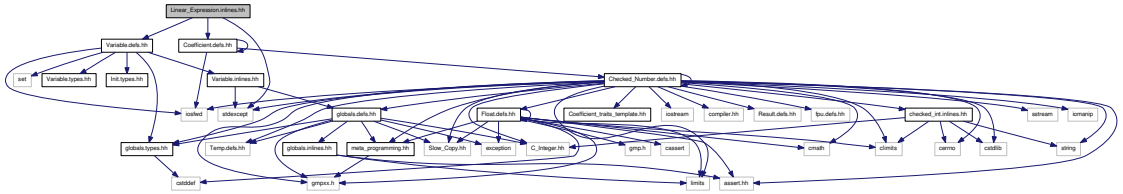
- namespace std

  The standard C++ namespace.

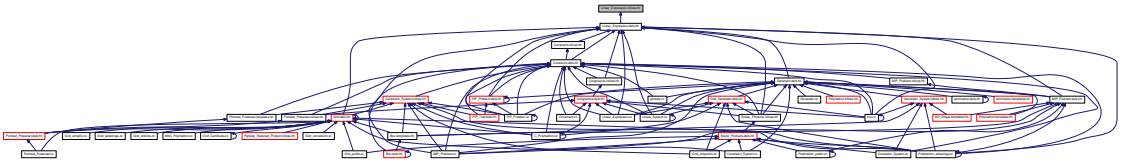
12.138 Linear_Expression.inlines.hh File Reference

```
#include "Variable.defs.hh"
#include "Coefficient.defs.hh"
#include <stdexcept>
```

Include dependency graph for Linear_Expression.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

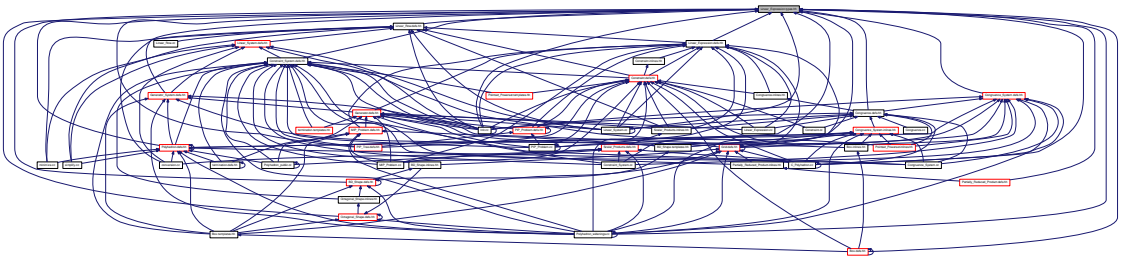
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.139 Linear_Expression.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

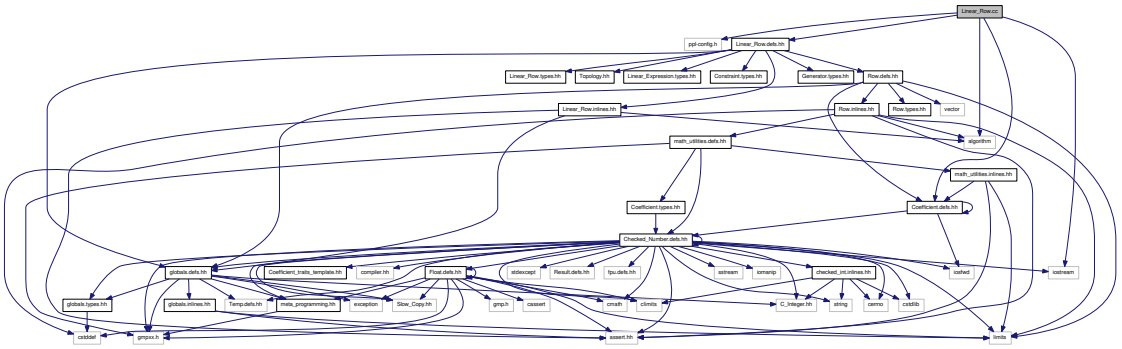
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.140 Linear_Row.cc File Reference

```
#include <ppl-config.h>
#include "Linear_Row.defs.hh"
#include "Coefficient.defs.hh"
#include <algorithm>
#include <iostream>
```

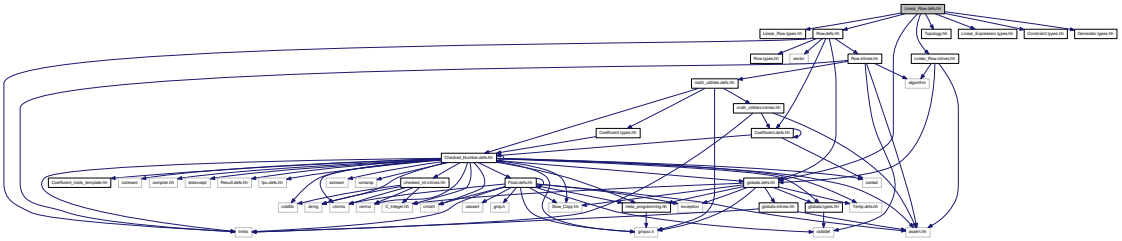
Include dependency graph for Linear_Row.cc:



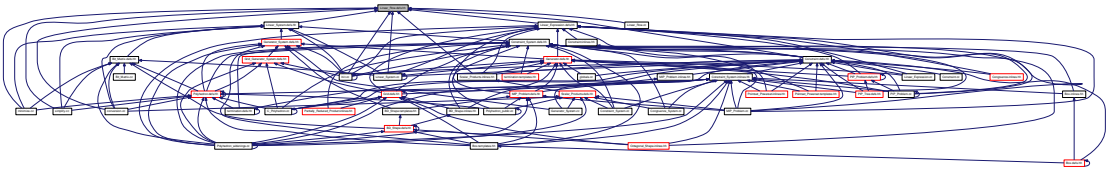
12.141 Linear_Row.defs.hh File Reference

```
#include "Linear_Row.types.hh"
#include "globals.defs.hh"
#include "Row.defs.hh"
#include "Topology.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Linear_Row.inlines.hh"
```

Include dependency graph for Linear_Row.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Linear_Row

  The base class for linear expressions, constraints and generators.

- class Parma_Polyhedra_Library::Linear_Row::Flags

  The type of the object to which the coefficients refer to, encoding both topology and kind.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

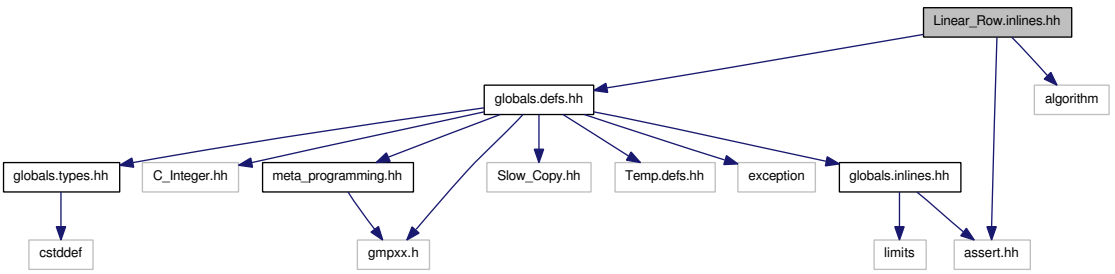
- namespace std

  The standard C++ namespace.

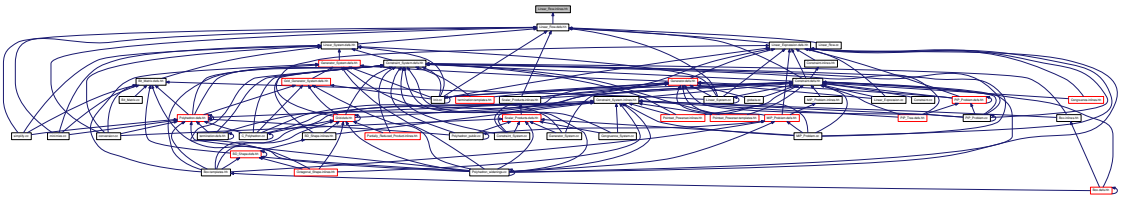
12.142 Linear_Row.inlines.hh File Reference

```
#include "globals.defs.hh"
#include "assert.hh"
#include <algorithm>
```

Include dependency graph for Linear_Row.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

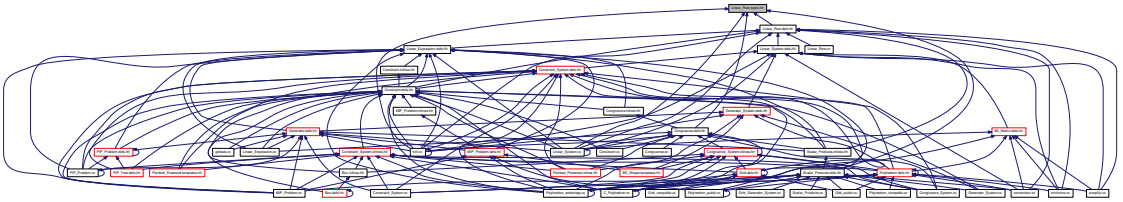
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.143 Linear_Row.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

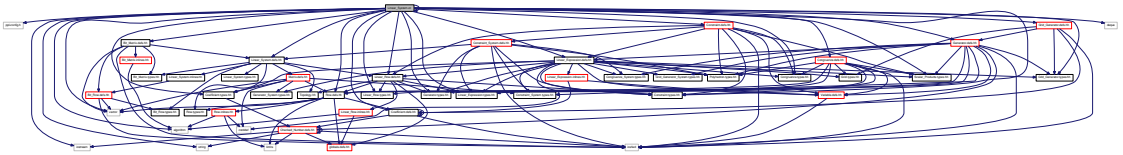
  The entire library is confined to this namespace.

12.144 Linear_System.cc File Reference

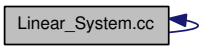
```
#include <ppl-config.h>
#include "Linear_System.defs.hh"
#include "Coefficient.defs.hh"
#include "Row.defs.hh"
#include "Bit_Matrix.defs.hh"
#include "Scalar_Products.defs.hh"
#include "Scalar_Products.types.hh"
#include "Linear_Row.types.hh"
```

```
#include "Coefficient.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Grid_Generator.types.hh"
#include "Linear_Row.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include "Generator.defs.hh"
#include "Congruence.defs.hh"
#include "Grid_Generator.defs.hh"
#include <algorithm>
#include <iostream>
#include <string>
#include <deque>
#include <vector>
```

Include dependency graph for Linear_System.cc:



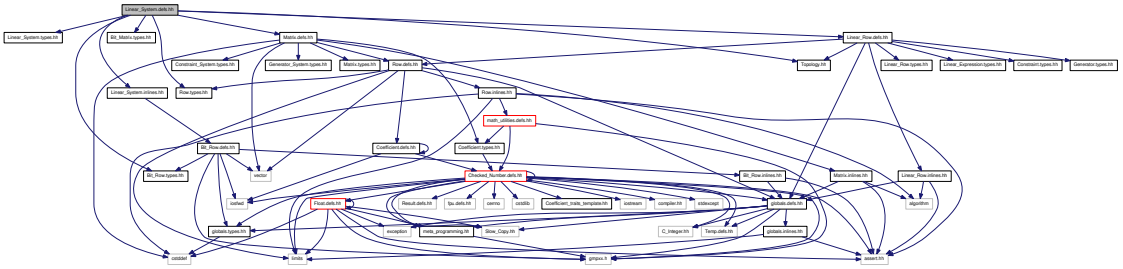
This graph shows which files directly or indirectly include this file:



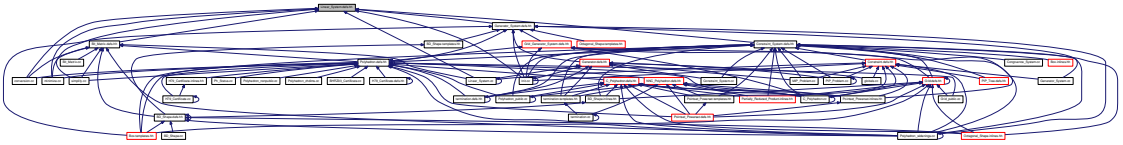
12.145 Linear_System.defs.hh File Reference

```
#include "Linear_System.types.hh"
#include "Row.types.hh"
#include "Bit_Row.types.hh"
#include "Bit_Matrix.types.hh"
#include "Matrix.defs.hh"
#include "Topology.hh"
#include "Linear_Row.defs.hh"
#include "Linear_System.inlines.hh"
```

Include dependency graph for Linear_System.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Linear_System

  The base class for systems of constraints and generators.

- struct Parma_Polyhedra_Library::Linear_System::With_Pending

  A tag class.

- struct Parma_Polyhedra_Library::Linear_System::Row_Less_Than

  Ordering predicate (used when implementing the sort algorithm).

Namespaces

- namespace std

  The standard C++ namespace.

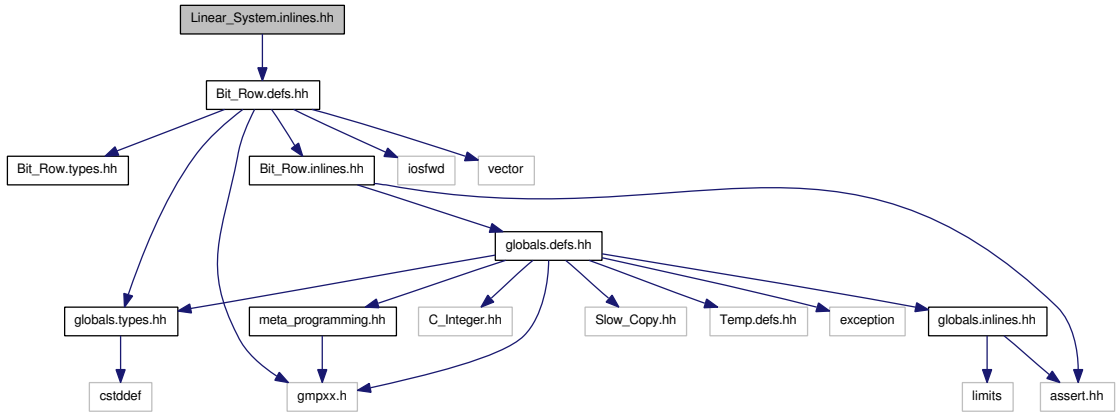
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

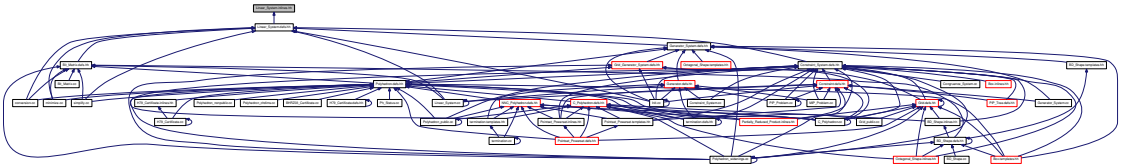
12.146 Linear_System.inlines.hh File Reference

```
#include "Bit_Row.defs.hh"
```

Include dependency graph for Linear_System.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

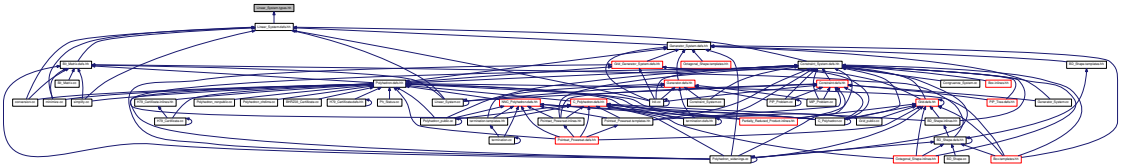
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.147 Linear_System.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

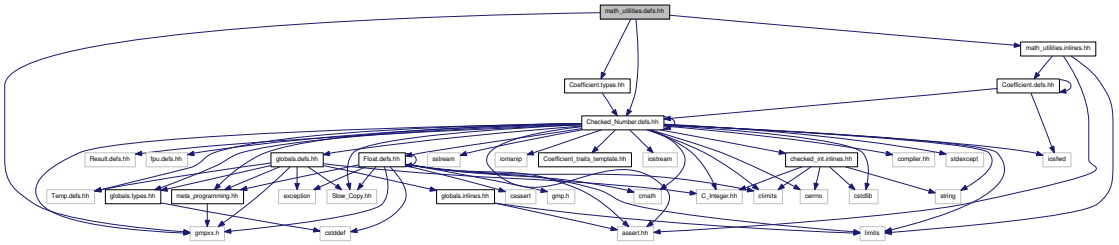
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

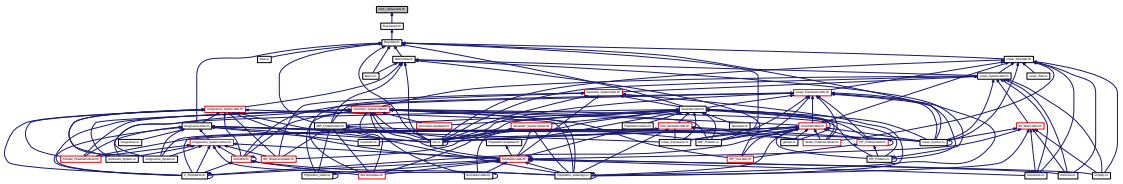
12.148 math_utilities.defs.hh File Reference

```
#include "Coefficient.types.hh"
#include "Checked_Number.defs.hh"
#include <gmpxx.h>
#include "math_utilities.inlines.hh"
```

Include dependency graph for math_utilities.defs.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::numer_denom (const T &from, Coefficient &num, Coefficient &den)

  Extract the numerator and denominator components of `from`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::div_round_up (T &to, Coefficient_traits::const_reference x, Coefficient_traits::const_reference y)

  Divides `x` by `y` into `to`, rounding the result towards plus infinity.

- template<typename N >
  void Parma_Polyhedra_Library::min_assign (N &x, const N &y)

  Assigns to `x` the minimum between `x` and `y`.

- template<typename N >
  void Parma_Polyhedra_Library::max_assign (N &x, const N &y)

  Assigns to `x` *the maximum between* `x` *and* `y`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_even (const T &x)

  Returns `true` *if and only if* `x` *is an even number.*

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_additive_inverse (const T &x, const T &y)

  Returns `true` *if and only if* $x = -y$.

- void Parma_Polyhedra_Library::normalize2 (Coefficient_traits::const_reference x, Coefficient_traits::const_reference y, Coefficient &nx, Coefficient &ny)

  If `g` *is the GCD of* `x` *and* `y`*, the values of* `x` *and* `y` *divided by* `g` *are assigned to* `nx` *and* `ny`*, respectively.*

- bool Parma_Polyhedra_Library::is_canonical (const mpq_class &x)

  Returns `true` *if and only if* `x` *is in canonical form.*

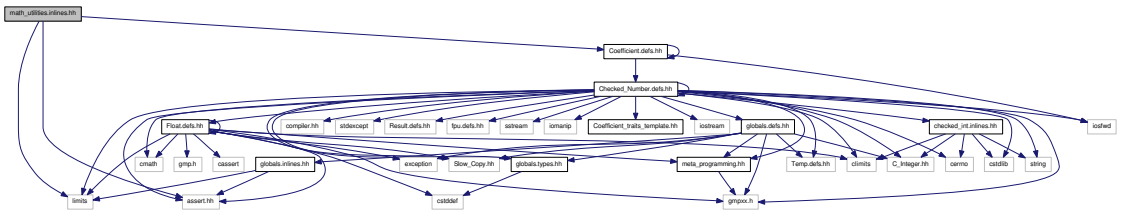
- template<typename T >
  T Parma_Polyhedra_Library::low_bits_mask (unsigned n)

  Returns a mask for the lowest `n` *bits,.*

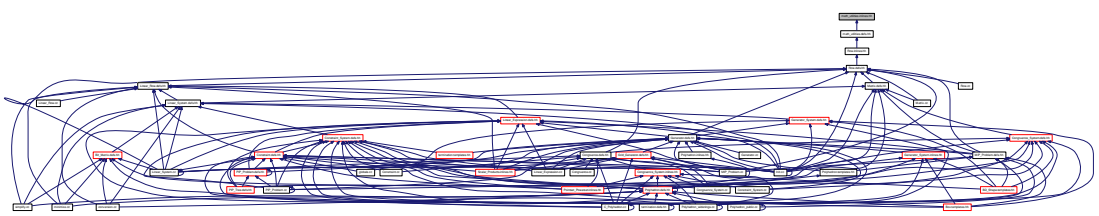
12.149 math_utilities.inlines.hh File Reference

```
#include "Coefficient.defs.hh"
#include <limits>
#include "assert.hh"
```

Include dependency graph for math_utilities.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- void Parma_Polyhedra_Library::normalize2 (Coefficient_traits::const_reference x, Coefficient_traits::const_reference y, Coefficient &nx, Coefficient &ny)

  If `g` is the GCD of `x` and `y`, the values of `x` and `y` divided by `g` are assigned to `nx` and `ny`, respectively.

- template<typename T >
  T Parma_Polyhedra_Library::low_bits_mask (unsigned n)

  Returns a mask for the lowest `n` bits,.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::numer_denom (const T &from, Coefficient &num, Coefficient &den)

  Extract the numerator and denominator components of `from`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, void >::type Parma_Polyhedra_Library::div_round_up (T &to, Coefficient_traits::const_reference x, Coefficient_traits::const_reference y)

  Divides `x` by `y` into `to`, rounding the result towards plus infinity.

- template<typename N >
  void Parma_Polyhedra_Library::min_assign (N &x, const N &y)

  Assigns to `x` the minimum between `x` and `y`.

- template<typename N >
  void Parma_Polyhedra_Library::max_assign (N &x, const N &y)

  Assigns to `x` the maximum between `x` and `y`.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_even (const T &x)

  Returns `true` if and only if `x` is an even number.

- template<typename T >
  Enable_If< Is_Native_Or_Checked< T >::value, bool >::type Parma_Polyhedra_Library::is_additive_inverse (const T &x, const T &y)

  Returns `true` if and only if $x = -y$.

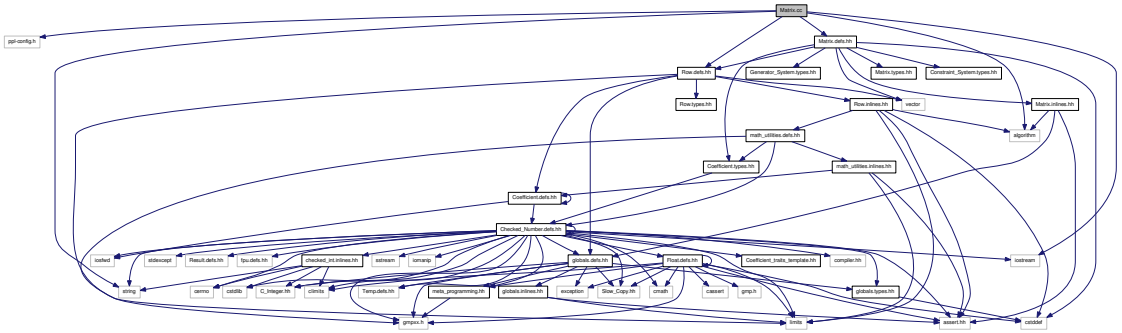
- bool Parma_Polyhedra_Library::is_canonical (const mpq_class &x)

  Returns `true` if and only if `x` is in canonical form.

12.150 Matrix.cc File Reference

```
#include <ppl-config.h>
#include "Matrix.defs.hh"
#include "Row.defs.hh"
#include <algorithm>
#include <iostream>
#include <string>
```

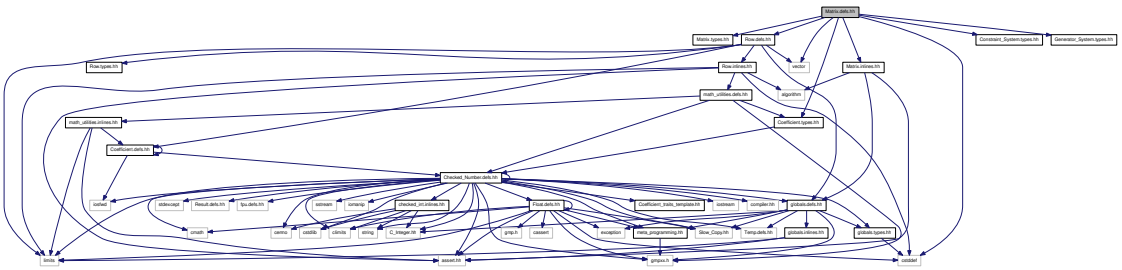
Include dependency graph for Matrix.cc:



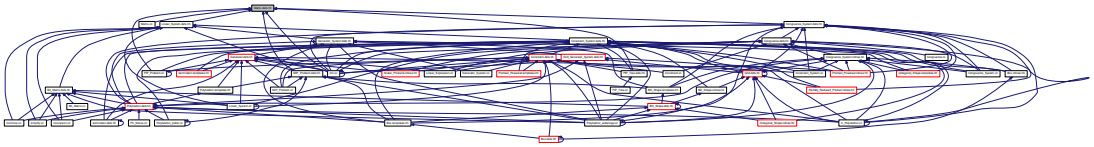
12.151 Matrix.defs.hh File Reference

```
#include "Matrix.types.hh"
#include "Row.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.types.hh"
#include "Coefficient.types.hh"
#include <vector>
#include <cstddef>
#include "Matrix.inlines.hh"
```

Include dependency graph for Matrix.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Matrix

  A 2-dimensional matrix of coefficients.

- class Parma_Polyhedra_Library::Matrix::const_iterator

  An iterator over a matrix.

Namespaces

- namespace std

  The standard C++ namespace.

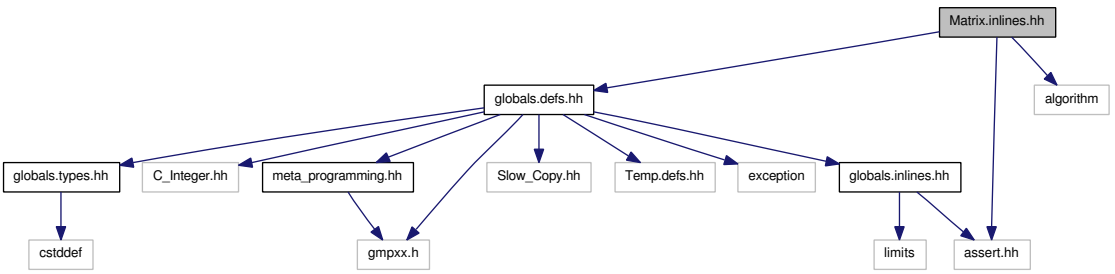
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

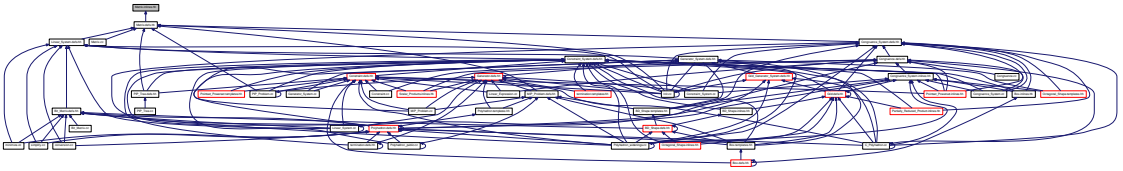
12.152 Matrix.inlines.hh File Reference

```
#include "globals.defs.hh"
#include <algorithm>
#include "assert.hh"
```

Include dependency graph for Matrix.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

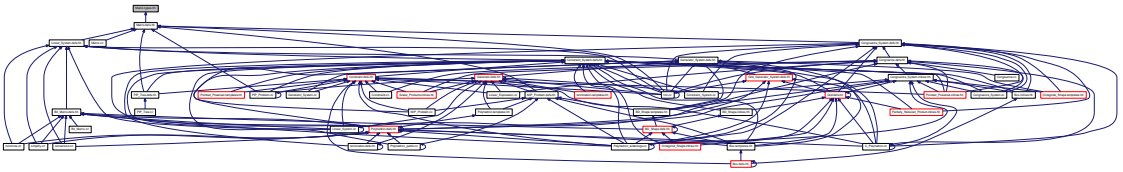
  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.153 Matrix.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

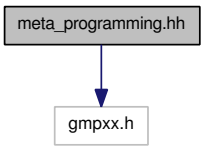
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

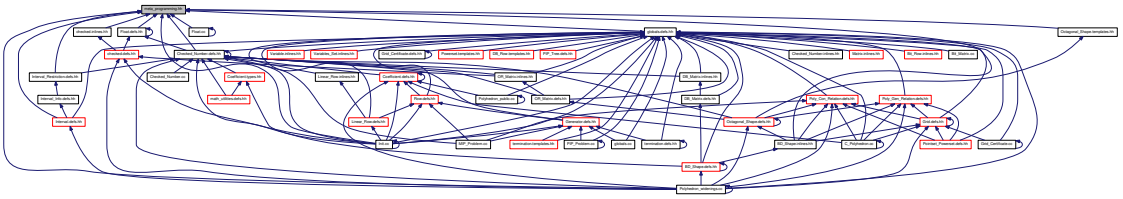
12.154 meta_programming.hh File Reference

`#include <gmpxx.h>`

Include dependency graph for meta_programming.hh:



This graph shows which files directly or indirectly include this file:



Classes

- struct Parma_Polyhedra_Library::Compile_Time_Check< true >

  A class that is only defined if `b` *evaluates to* `true`.

- struct Parma_Polyhedra_Library::Bool< b >

  A class holding a constant called `value` *that evaluates to* `b`.

- struct Parma_Polyhedra_Library::True

  A class holding a constant called `value` *that evaluates to* `true`.

- struct Parma_Polyhedra_Library::False

  A class holding a constant called `value` *that evaluates to* `false`.

- struct Parma_Polyhedra_Library::Is_Same< T1, T2 >

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `T1` *is the same type as* `T2`.

- struct Parma_Polyhedra_Library::Is_Same< T, T >

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `T1` *is the same type as* `T2`.

- struct Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >

  A class holding a constant called `value` *that evaluates to* `true` *if and only if* `Base` *is the same type as* `Derived` *or* `Derived` *is a class derived from* `Base`.

- struct Parma_Polyhedra_Library::Is_Same_Or_Derived< Base, Derived >::Any

  A class that is constructible from just anything.

- struct Parma_Polyhedra_Library::Enable_If< b, T >

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`.

- struct Parma_Polyhedra_Library::Enable_If_Is< Type, Type, T >
- struct Parma_Polyhedra_Library::Enable_If< true, T >

  A class that provides a type member called `type` *equivalent to* `T` *if and only if* `b` *is* `true`.

- struct Parma_Polyhedra_Library::Is_Native< T >
- struct Parma_Polyhedra_Library::Is_Native< char >
- struct Parma_Polyhedra_Library::Is_Native< signed char >
- struct Parma_Polyhedra_Library::Is_Native< signed short >
- struct Parma_Polyhedra_Library::Is_Native< signed int >
- struct Parma_Polyhedra_Library::Is_Native< signed long >

- struct Parma_Polyhedra_Library::Is_Native< signed long long >
- struct Parma_Polyhedra_Library::Is_Native< unsigned char >
- struct Parma_Polyhedra_Library::Is_Native< unsigned short >
- struct Parma_Polyhedra_Library::Is_Native< unsigned int >
- struct Parma_Polyhedra_Library::Is_Native< unsigned long >
- struct Parma_Polyhedra_Library::Is_Native< unsigned long long >
- struct Parma_Polyhedra_Library::Is_Native< mpz_class >
- struct Parma_Polyhedra_Library::Is_Native< mpq_class >

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Defines

- #define const_bool_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

  Declares a per-class constant of type `bool`, called `name` and with value `value`.

- #define const_int_nodef(name, value) enum anonymous_enum_ ## name { name = (value) }

  Declares a per-class constant of type `int`, called `name` and with value `value`.

- #define const_value_nodef(type, name, value)

  Declares a per-class constant of type `type`, called `name` and with value `value`. The value of the constant is accessible by means of the syntax `name()`.

- #define const_ref_nodef(type, name, value)

  Declares a per-class constant of type `type`, called `name` and with value `value`. A constant reference to the constant is accessible by means of the syntax `name()`.

- #define PPL_COMPILE_TIME_CHECK_NAME(suffix) compile_time_check_ ## suffix
- #define PPL_COMPILE_TIME_CHECK_AUX(e, suffix)
- #define PPL_COMPILE_TIME_CHECK(e, msg) PPL_COMPILE_TIME_CHECK_AUX(e, __LINE__)

  Produces a compilation error if the compile-time constant `e` does not evaluate to `true`

12.154.1 Define Documentation

12.154.1.1 #define PPL_COMPILE_TIME_CHECK_AUX(e, suffix)

Definition at line 119 of file meta_programming.hh.

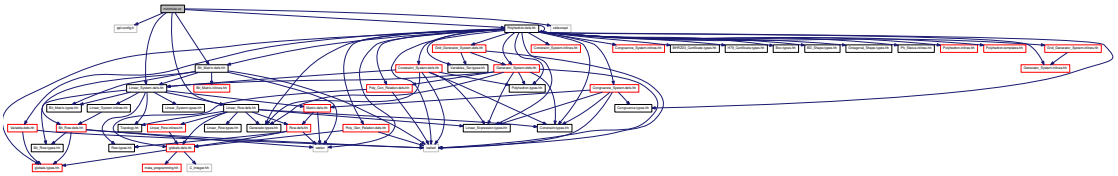
12.154.1.2 #define PPL_COMPILE_TIME_CHECK_NAME(suffix) compile_time_check_ ## suffix

Definition at line 118 of file meta_programming.hh.

12.155 minimize.cc File Reference

```
#include <ppl-config.h>
#include "Linear_Row.defs.hh"
#include "Linear_System.defs.hh"
#include "Bit_Matrix.defs.hh"
#include "Polyhedron.defs.hh"
#include <stdexcept>
```

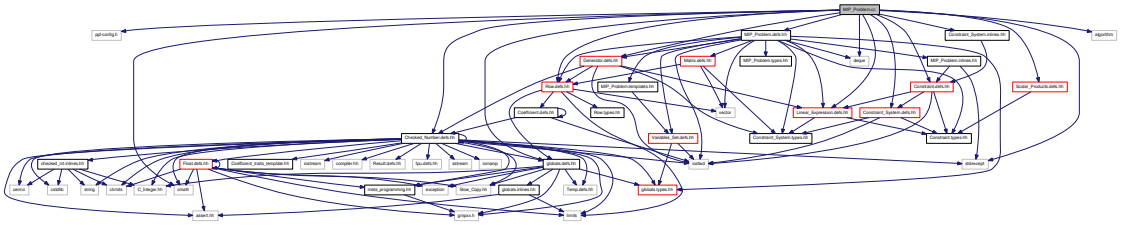
Include dependency graph for minimize.cc:



12.156 MIP_Problem.cc File Reference

```
#include <ppl-config.h>
#include "MIP_Problem.defs.hh"
#include "globals.defs.hh"
#include "Checked_Number.defs.hh"
#include "Row.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Generator.defs.hh"
#include "Scalar_Products.defs.hh"
#include <stdexcept>
#include <deque>
#include <algorithm>
#include <cmath>
```

Include dependency graph for MIP_Problem.cc:



Defines

- #define PPL_SIMPLEX_USE_MIP_HEURISTIC 1

12.156.1 Define Documentation

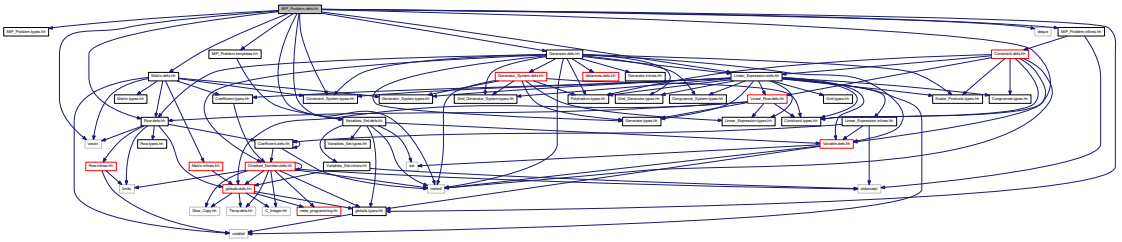
12.156.1.1 #define PPL_SIMPLEX_USE_MIP_HEURISTIC 1

Definition at line 45 of file MIP_Problem.cc.

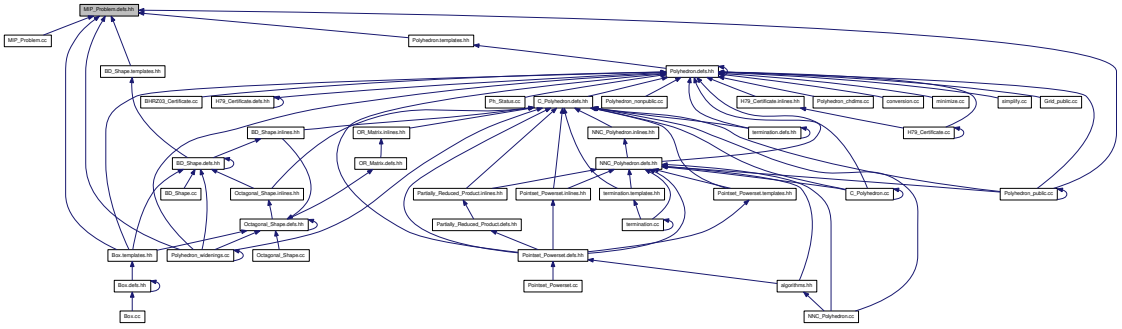
12.157 MIP_Problem.defs.hh File Reference

```
#include "MIP_Problem.types.hh"
#include "globals.types.hh"
#include "Row.defs.hh"
#include "Matrix.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.types.hh"
#include "Constraint_System.types.hh"
#include "Generator.defs.hh"
#include "Variables_Set.defs.hh"
#include <vector>
#include <deque>
#include <iosfwd>
#include "MIP_Problem.inlines.hh"
#include "MIP_Problem.templates.hh"
```

Include dependency graph for MIP_Problem.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::MIP_Problem

  A Mixed Integer (linear) Programming problem.

- struct Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation

  A helper class to temporarily relax a MIP problem using RAII.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

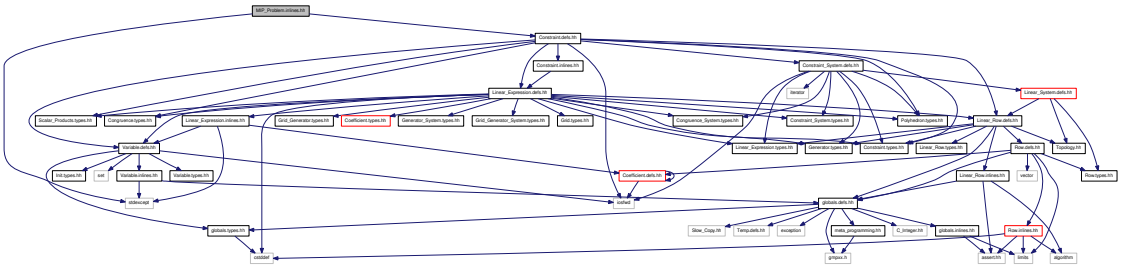
  The standard C++ namespace.

12.158 MIP_Problem.inlines.hh File Reference

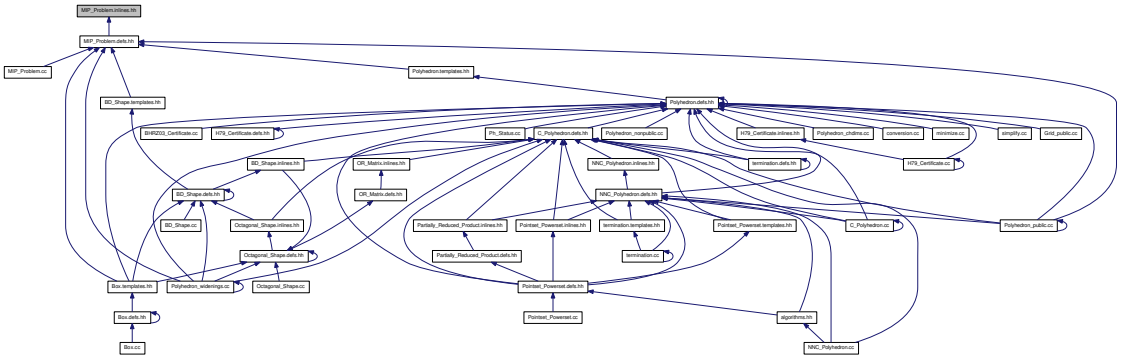
```
#include "Constraint.defs.hh"
```

```
#include <stdexcept>
```

Include dependency graph for MIP_Problem.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

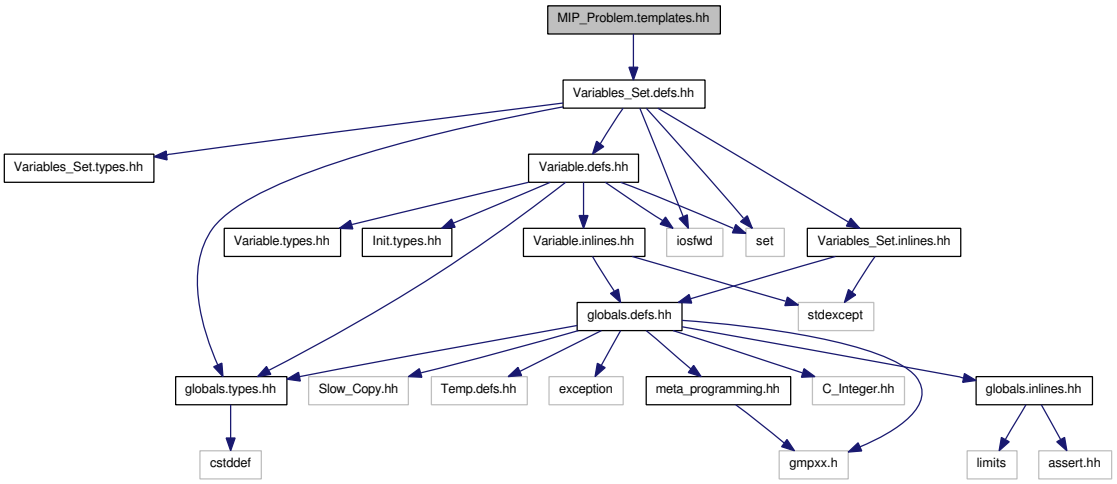
- namespace std

  The standard C++ namespace.

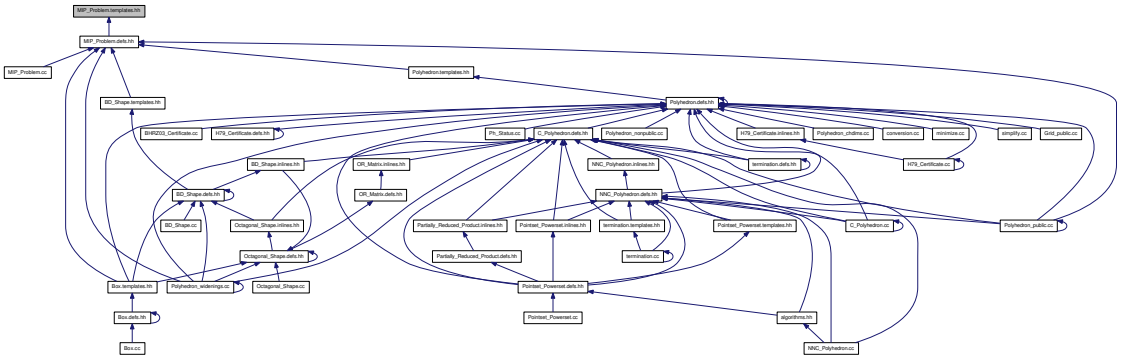
12.159 MIP_Problem.templates.hh File Reference

```
#include "Variables_Set.defs.hh"
```

Include dependency graph for MIP_Problem.templates.hh:



This graph shows which files directly or indirectly include this file:



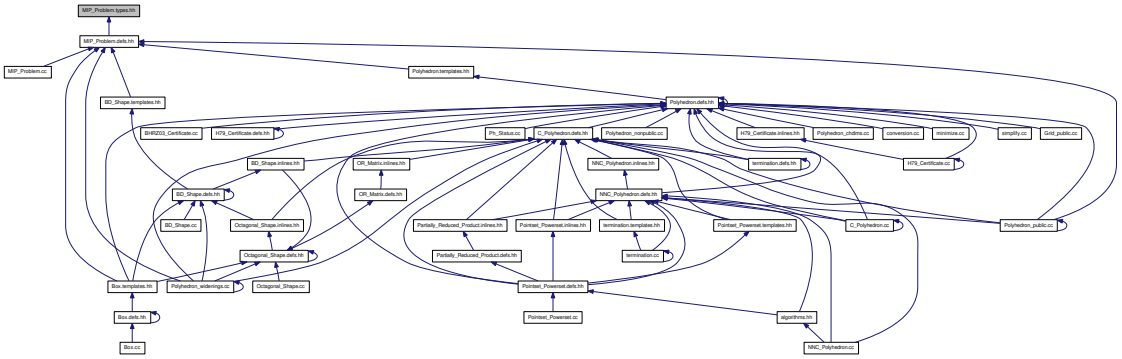
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.160 MIP_Problem.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Enumerations

- enum Parma_Polyhedra_Library::MIP_Problem_Status { Parma_Polyhedra_Library::UNFEASIBLE_MIP_PROBLEM, Parma_Polyhedra_Library::UNBOUNDED_MIP_PROBLEM, Parma_Polyhedra_Library::OPTIMIZED_MIP_PROBLEM }

  Possible outcomes of the MIP_Problem solver.

12.161 namespaces.hh File Reference

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace Parma_Polyhedra_Library::Checked

  Types and functions implementing checked numbers.

- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

- namespace Parma_Polyhedra_Library::Interfaces

  Data and functions related to language interfaces.

- namespace Parma_Polyhedra_Library::Interfaces::C

  Data and functions related to the C language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Java

  Data and functions related to the Java language interface.

- namespace Parma_Polyhedra_Library::Interfaces::OCaml

  Data and functions related to the OCaml language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog

  Data and functions related to the Prolog language interfaces.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::Ciao

  Data and functions related to the Ciao Prolog language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::GNU

  Data and functions related to the GNU Prolog language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::SICStus

  Data and functions related to the SICStus language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::SWI

  Data and functions related to the SWI-Prolog language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::XSB

  Data and functions related to the XSB language interface.

- namespace Parma_Polyhedra_Library::Interfaces::Prolog::YAP

  Data and functions related to the YAP language interface.

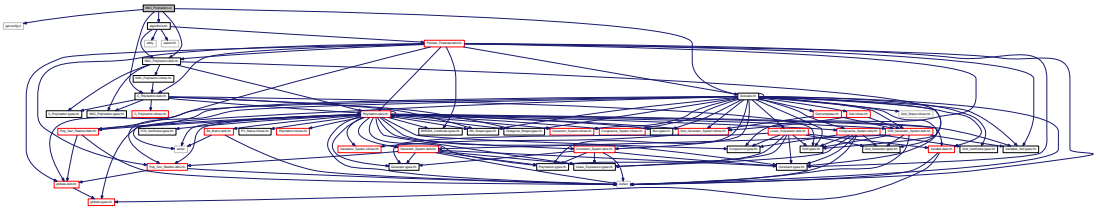
- namespace std

  The standard C++ namespace.

12.162 NNC_Polyhedron.cc File Reference

```
#include <ppl-config.h>
#include "NNC_Polyhedron.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "algorithms.hh"
```

Include dependency graph for NNC_Polyhedron.cc:



Defines

- #define USE_BHZ09 1

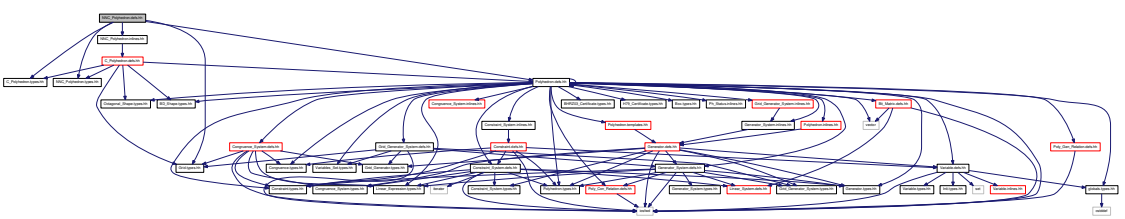
12.162.1 Define Documentation

12.162.1.1 #define USE_BHZ09 1

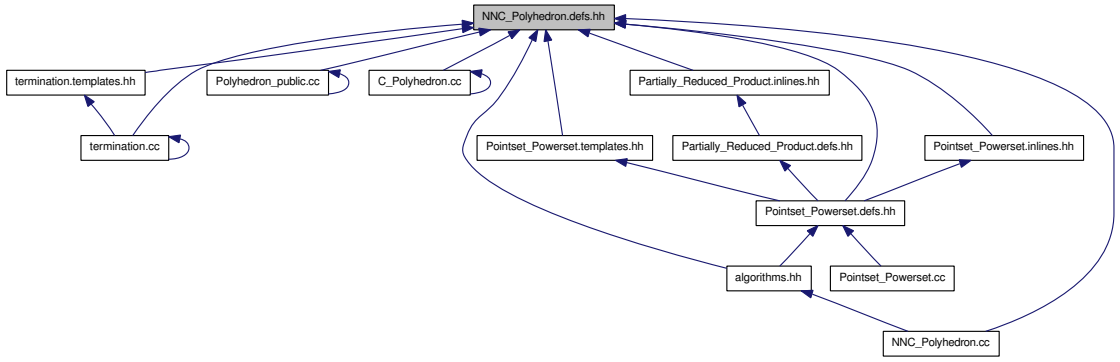
12.163 NNC_Polyhedron.defs.hh File Reference

```
#include "C_Polyhedron.types.hh"
#include "NNC_Polyhedron.types.hh"
#include "Polyhedron.defs.hh"
#include "Grid.types.hh"
#include "NNC_Polyhedron.inlines.hh"
```

Include dependency graph for NNC_Polyhedron.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

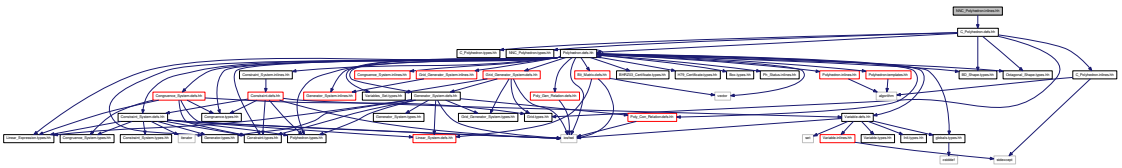
- class Parma_Polyhedra_Library::NNC_Polyhedron

  A not necessarily closed convex polyhedron.

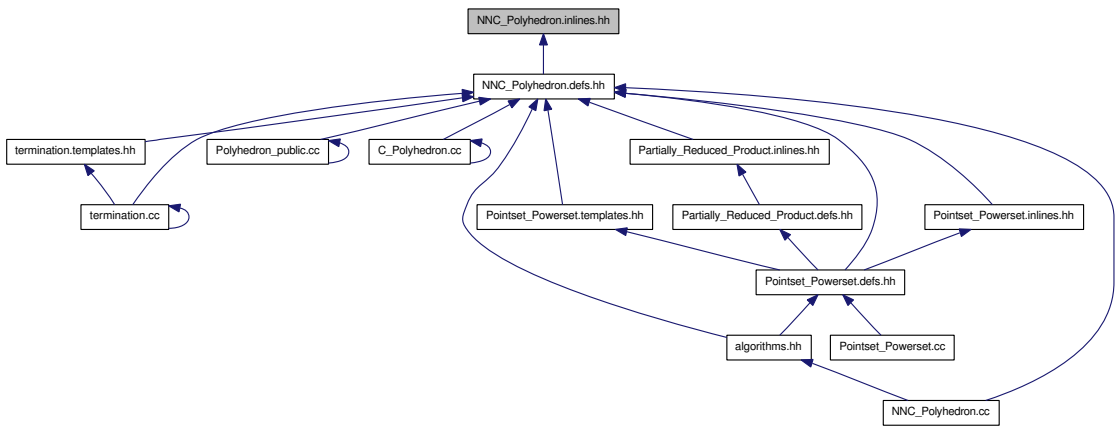
12.164 NNC_Polyhedron.inlines.hh File Reference

```
#include "C_Polyhedron.defs.hh"
```

Include dependency graph for NNC_Polyhedron.inlines.hh:



This graph shows which files directly or indirectly include this file:



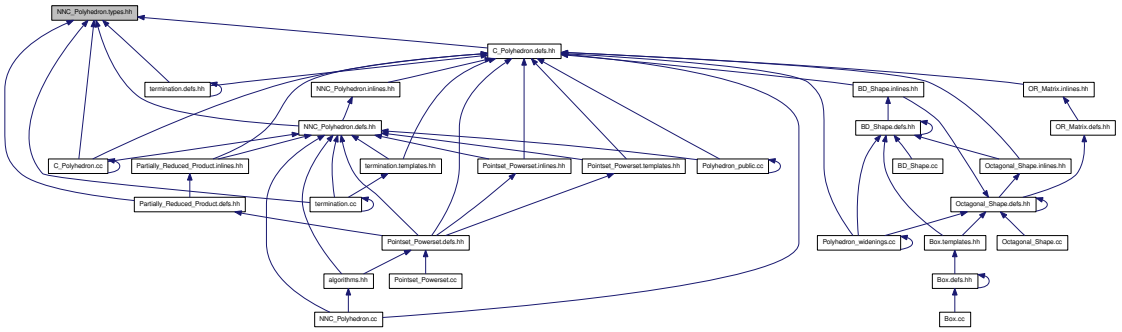
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.165 NNC_Polyhedron.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

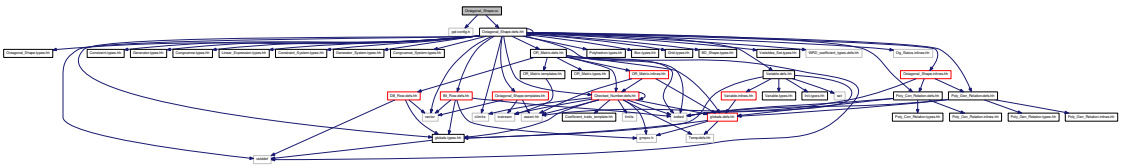
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.166 Octagonal_Shape.cc File Reference

```
#include <ppl-config.h>
#include "Octagonal_Shape.defs.hh"
```

Include dependency graph for Octagonal_Shape.cc:

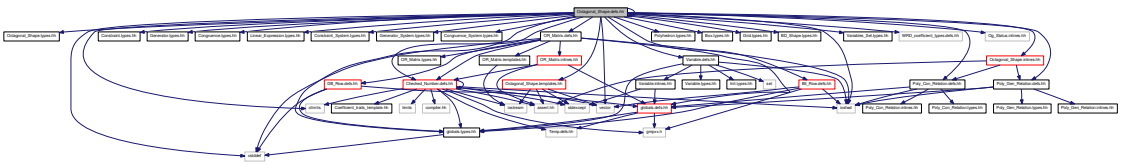


12.167 Octagonal_Shape.defs.hh File Reference

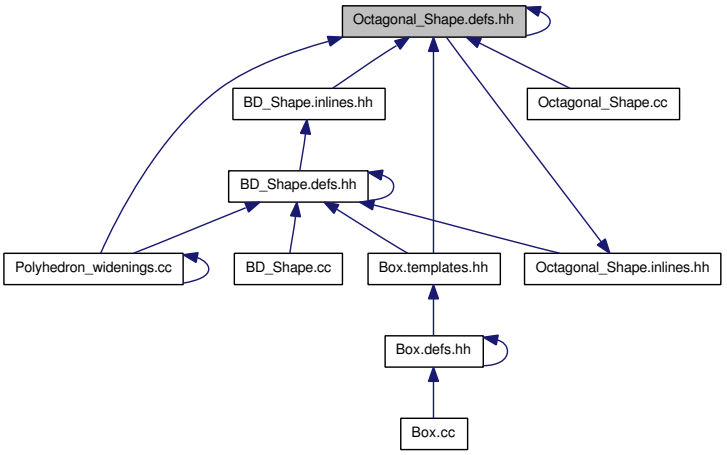
```
#include "Octagonal_Shape.types.hh"
#include "globals.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
```

```
#include "Congruence.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.types.hh"
#include "Congruence_System.types.hh"
#include "OR_Matrix.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "Polyhedron.types.hh"
#include "Box.types.hh"
#include "Grid.types.hh"
#include "BD_Shape.types.hh"
#include "Variable.defs.hh"
#include "Variables_Set.types.hh"
#include "Checked_Number.defs.hh"
#include "WRD_coefficient_types.defs.hh"
#include "Bit_Row.defs.hh"
#include <vector>
#include <cstddef>
#include <climits>
#include <iosfwd>
#include "Og_Status.idefs.hh"
#include "Og_Status.inlines.hh"
#include "Octagonal_Shape.inlines.hh"
#include "Octagonal_Shape.templates.hh"
```

Include dependency graph for Octagonal_Shape.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Octagonal_Shape< T >

  An octagonal shape.

- class Parma_Polyhedra_Library::Octagonal_Shape< T >::Status

  A conjunctive assertion about a Octagonal_Shape<T> object.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_IN_Octagonal_Shape_CLASS

12.167.1 Define Documentation

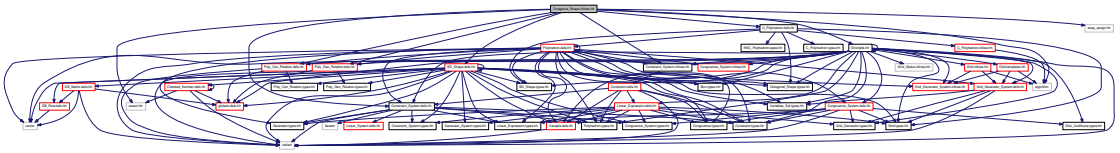
12.167.1.1 #define PPL_IN_Octagonal_Shape_CLASS

Definition at line 1791 of file Octagonal_Shape.defs.hh.

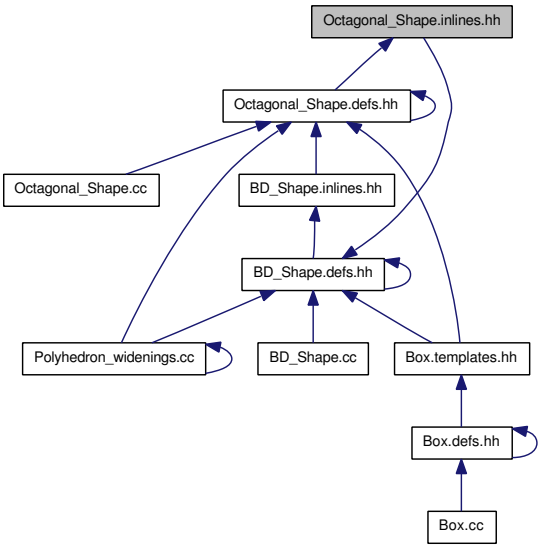
12.168 Octagonal_Shape.inlines.hh File Reference

```
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "C_Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "BD_Shape.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "wrap_assign.hh"
#include "assert.hh"
#include <algorithm>
```

Include dependency graph for Octagonal_Shape.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Implementation

Implementation related data and functions.

- namespace Parma_Polyhedra_Library::Implementation::Octagonal_Shapes

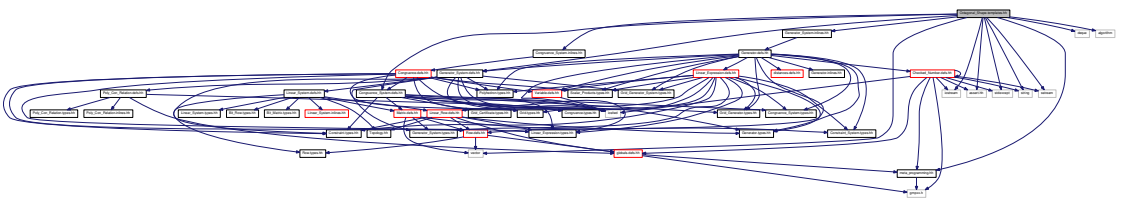
- namespace std

The standard C++ namespace.

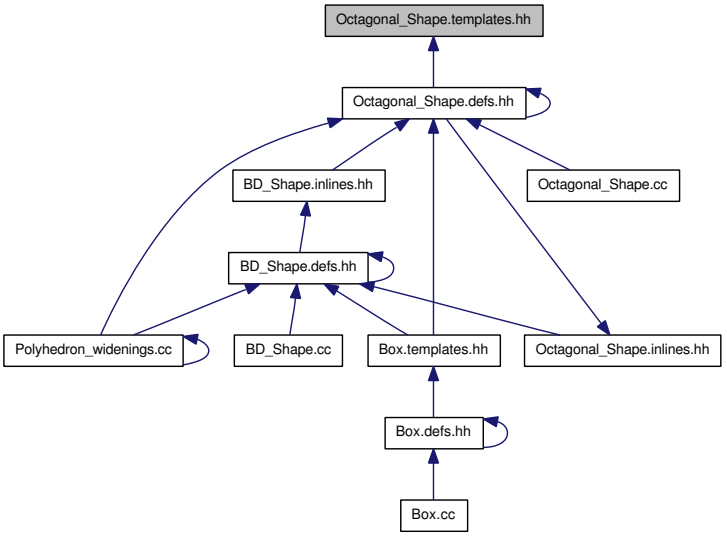
12.169 Octagonal_Shape.templates.hh File Reference

```
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "meta_programming.hh"
#include "assert.hh"
#include <vector>
#include <deque>
#include <string>
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <algorithm>
```

Include dependency graph for Octagonal_Shape.templates.hh:



This graph shows which files directly or indirectly include this file:



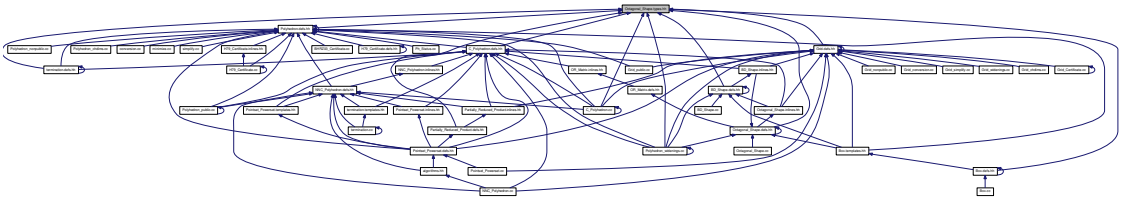
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.170 Octagonal_Shape.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

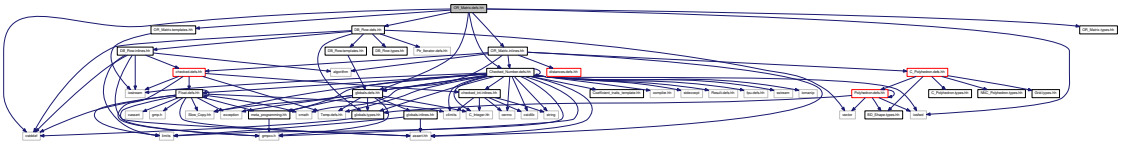
  The entire library is confined to this namespace.

12.171 OR_Matrix.defs.hh File Reference

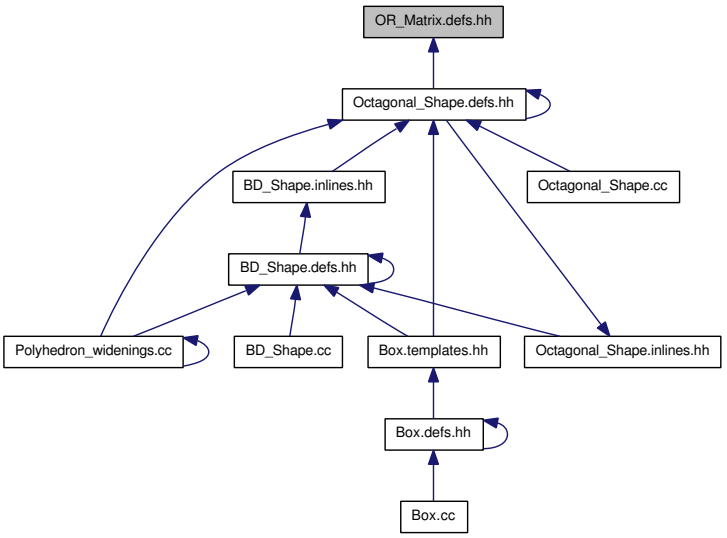
```
#include "globals.defs.hh"
#include "OR_Matrix.types.hh"
#include "DB_Row.defs.hh"
```

```
#include "Checked_Number.defs.hh"
#include <cstddef>
#include <iosfwd>
#include "OR_Matrix.inlines.hh"
#include "OR_Matrix.templates.hh"
```

Include dependency graph for OR_Matrix.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::OR_Matrix< T >

  A matrix representing octagonal constraints.

- class Parma_Polyhedra_Library::OR_Matrix< T >::Pseudo_Row< U >

  An object that behaves like a matrix's row with respect to the subscript operators.

- class Parma_Polyhedra_Library::OR_Matrix< T >::any_row_iterator< U >

  A template class to derive both OR_Matrix::iterator and OR_Matrix::const_iterator.

Namespaces

- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_OR_MATRIX_EXTRA_DEBUG 0

12.171.1 Define Documentation

12.171.1.1 #define PPL_OR_MATRIX_EXTRA_DEBUG 0

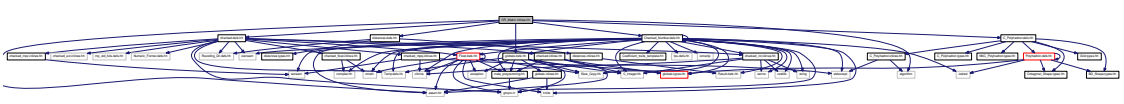
Definition at line 47 of file OR_Matrix.defs.hh.

Referenced by Parma_Polyhedra_Library::OR_Matrix< T >::operator[ ]().

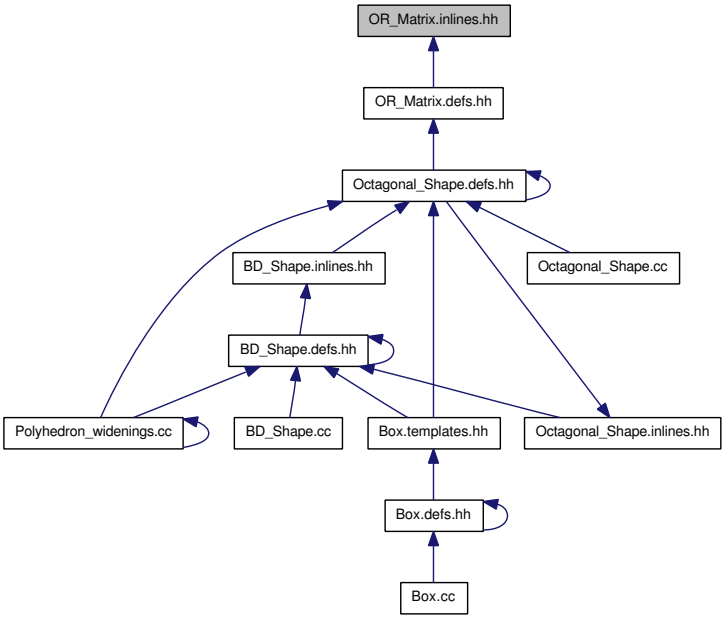
12.172 OR_Matrix.inlines.hh File Reference

```
#include "globals.defs.hh"
#include "Checked_Number.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "distances.defs.hh"
#include "assert.hh"
#include <algorithm>
#include "checked.defs.hh"
```

Include dependency graph for OR_Matrix.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

Functions

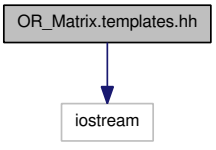
- unsigned long Parma_Polyhedra_Library::isqrt (unsigned long x)

  Returns the integer square root of `x`.

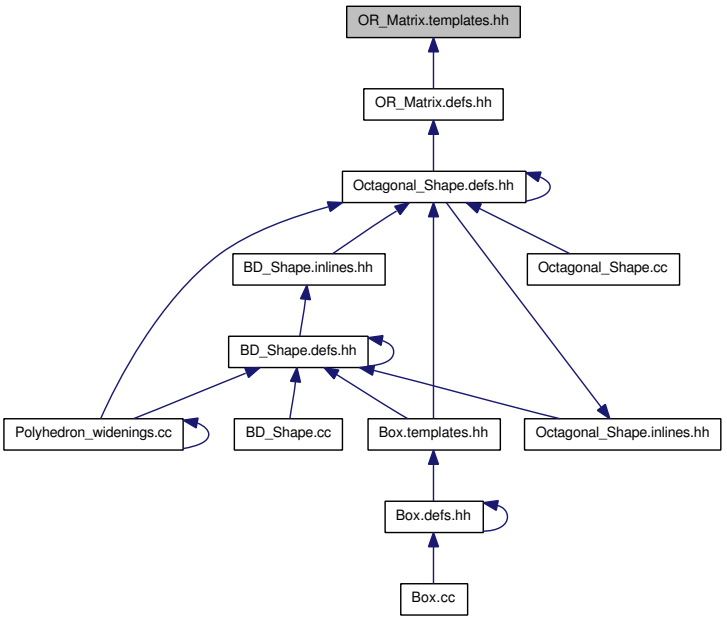
12.173 OR_Matrix.templates.hh File Reference

```
#include <iostream>
```

Include dependency graph for OR_Matrix.templates.hh:



This graph shows which files directly or indirectly include this file:



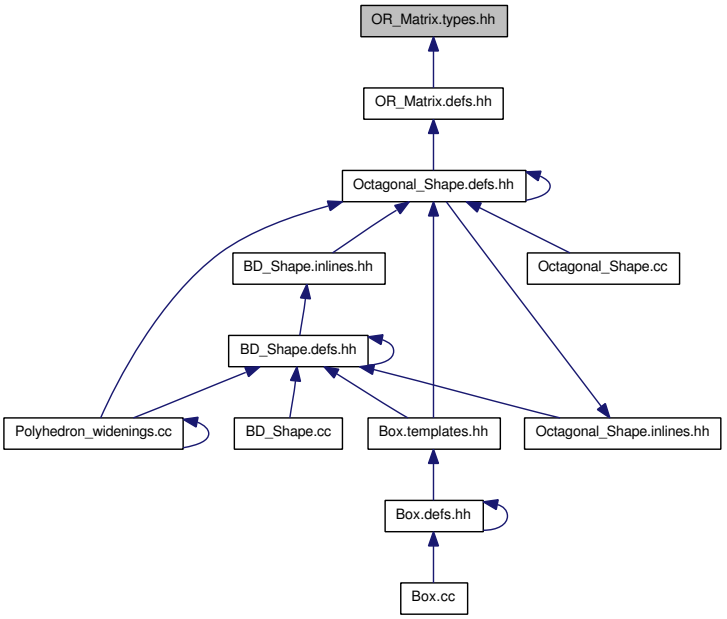
Namespaces

- namespace Parma_Polyhedra_Library

The entire library is confined to this namespace.

12.174 OR_Matrix.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

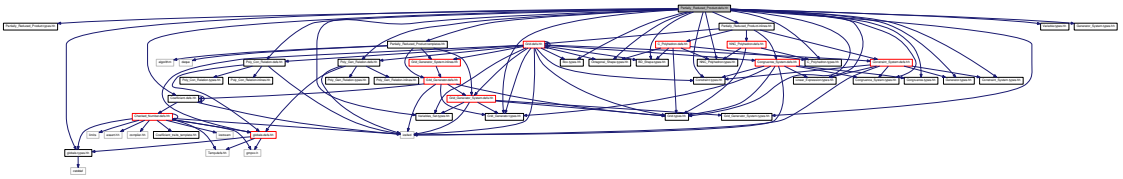
  The entire library is confined to this namespace.

12.175 Partially_Reduced_Product.defs.hh File Reference

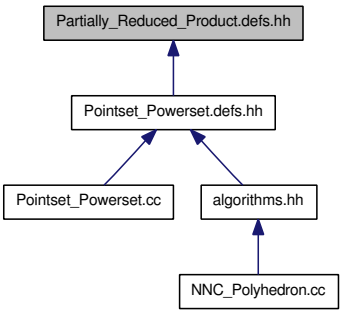
```
#include "Partially_Reduced_Product.types.hh"
#include "globals.types.hh"
#include "Coefficient.defs.hh"
#include "Variable.types.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Grid_Generator.types.hh"
#include "Constraint_System.types.hh"
#include "Generator_System.types.hh"
#include "Congruence_System.types.hh"
```

```
#include "Grid_Generator_System.types.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "C_Polyhedron.types.hh"
#include "NNC_Polyhedron.types.hh"
#include "Grid.types.hh"
#include "Box.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include "Partially_Reduced_Product.inlines.hh"
#include "Partially_Reduced_Product.templates.hh"
```

Include dependency graph for Partially_Reduced_Product.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Smash_Reduction< D1, D2 >

  This class provides the reduction method for the Smash_Product domain.

- class Parma_Polyhedra_Library::Constraints_Reduction< D1, D2 >

  This class provides the reduction method for the Constraints_Product domain.

- class Parma_Polyhedra_Library::Congruences_Reduction< D1, D2 >

  This class provides the reduction method for the Congruences_Product domain.

- class Parma_Polyhedra_Library::Shape_Preserving_Reduction< D1, D2 >

This class provides the reduction method for the Shape_Preserving_Product domain.

- class Parma_Polyhedra_Library::No_Reduction< D1, D2 >

  This class provides the reduction method for the Direct_Product domain.

- class Parma_Polyhedra_Library::Partially_Reduced_Product< D1, D2, R >

  The partially reduced product of two abstractions.

- class Parma_Polyhedra_Library::Domain_Product< D1, D2 >

  This class is temporary and will be removed when template typedefs will be supported in C++.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

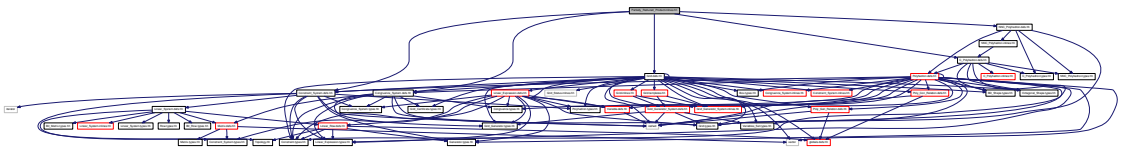
- namespace std

  The standard C++ namespace.

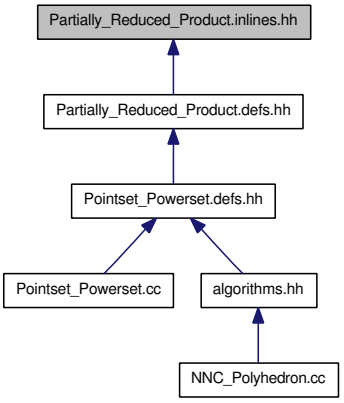
12.176 Partially_Reduced_Product.inlines.hh File Reference

```
#include "Constraint_System.defs.hh"
#include "Congruence_System.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include "Grid.defs.hh"
```

Include dependency graph for Partially_Reduced_Product.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

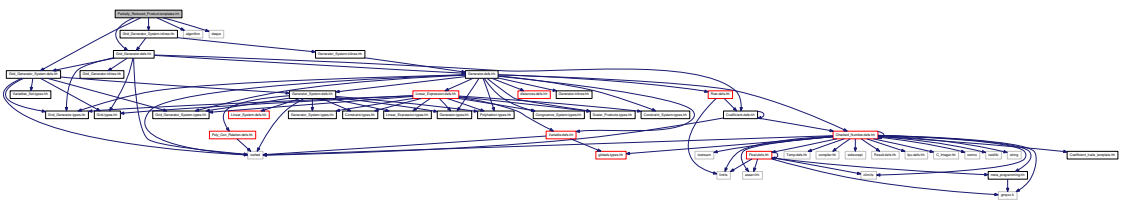
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

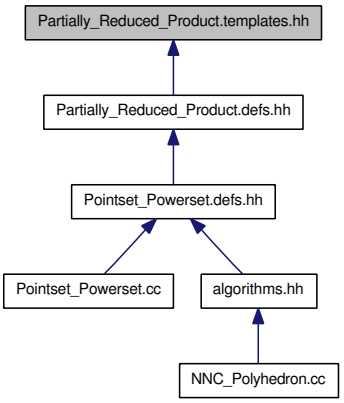
12.177 Partially_Reduced_Product.templates.hh File Reference

```
#include "Grid_Generator.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include <algorithm>
#include <deque>
```

Include dependency graph for Partially_Reduced_Product.templates.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

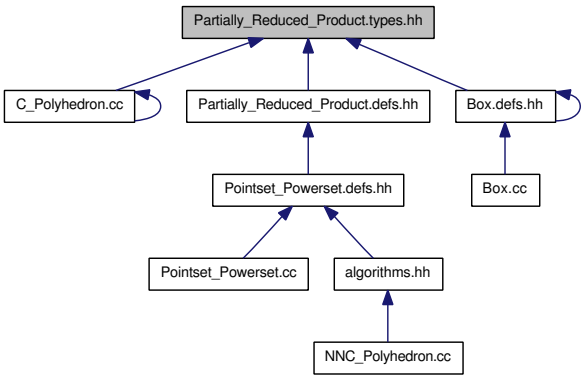
- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

Functions

- template<typename D1 , typename D2 >
  bool Parma_Polyhedra_Library::shrink_to_congruence_no_check (D1 &d1, D2 &d2, const Congruence &cg)

12.178 Partially_Reduced_Product.types.hh File Reference

This graph shows which files directly or indirectly include this file:



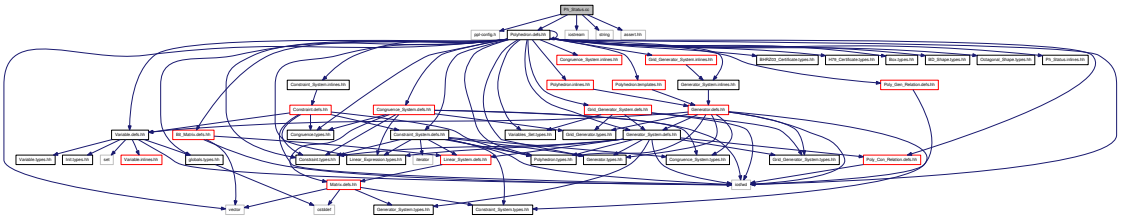
Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

12.179 Ph_Status.cc File Reference

```
#include <ppl-config.h>
#include "Polyhedron.defs.hh"
#include <iostream>
#include <string>
#include "assert.hh"
```

Include dependency graph for Ph_Status.cc:



12.180 Ph_Status.idefs.hh File Reference

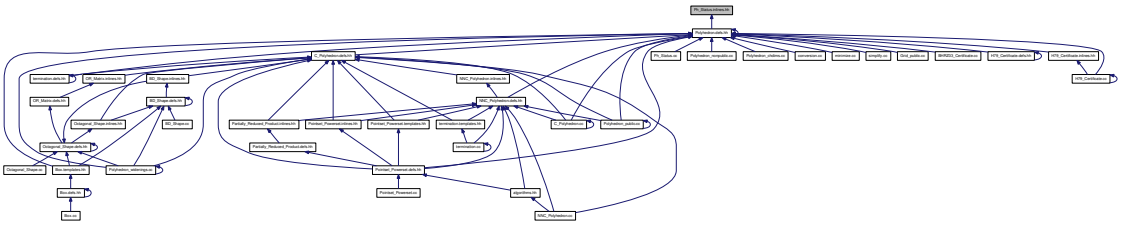
Classes

- class Status

  A conjunctive assertion about a polyhedron.

12.181 Ph_Status.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

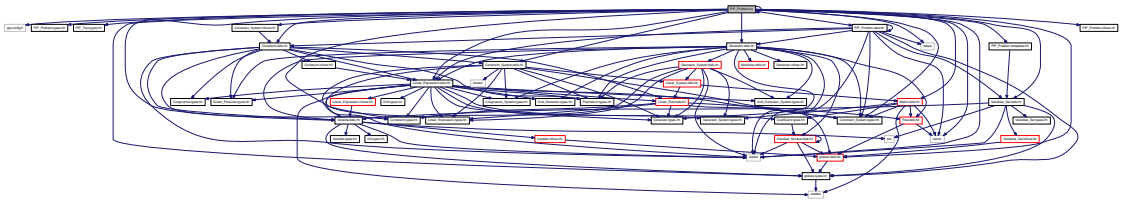
  The entire library is confined to this namespace.

12.182 PIP_Problem.cc File Reference

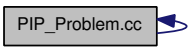
```
#include <ppl-config.h>
```

```
#include "PIP_Problem.defs.hh"
#include "PIP_Problem.types.hh"
#include "PIP_Tree.types.hh"
#include "globals.types.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator.defs.hh"
#include "Variables_Set.defs.hh"
#include <vector>
#include <deque>
#include <iosfwd>
#include "PIP_Problem.inlines.hh"
#include "PIP_Problem.templates.hh"
#include "Variable.defs.hh"
#include "Linear_Expression.types.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Matrix.defs.hh"
#include "globals.defs.hh"
#include "PIP_Problem.defs.hh"
```

Include dependency graph for PIP_Problem.cc:



This graph shows which files directly or indirectly include this file:

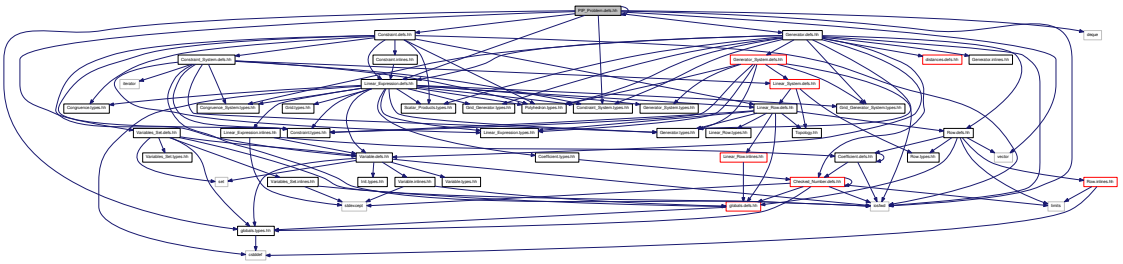


12.183 PIP_Problem.defs.hh File Reference

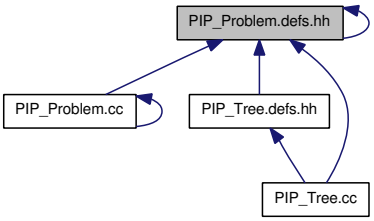
```
#include "PIP_Problem.types.hh"
#include "globals.types.hh"
#include "Linear_Expression.defs.hh"
```

```
#include "Constraint.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator.defs.hh"
#include "Variables_Set.defs.hh"
#include <vector>
#include <deque>
#include <iosfwd>
```

Include dependency graph for PIP_Problem.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::PIP_Problem

  A Parametric Integer (linear) Programming problem.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

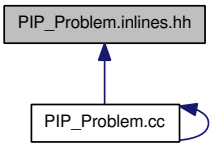
  All input/output operators are confined to this namespace.

- namespace std

  The standard C++ namespace.

12.184 PIP_Problem.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

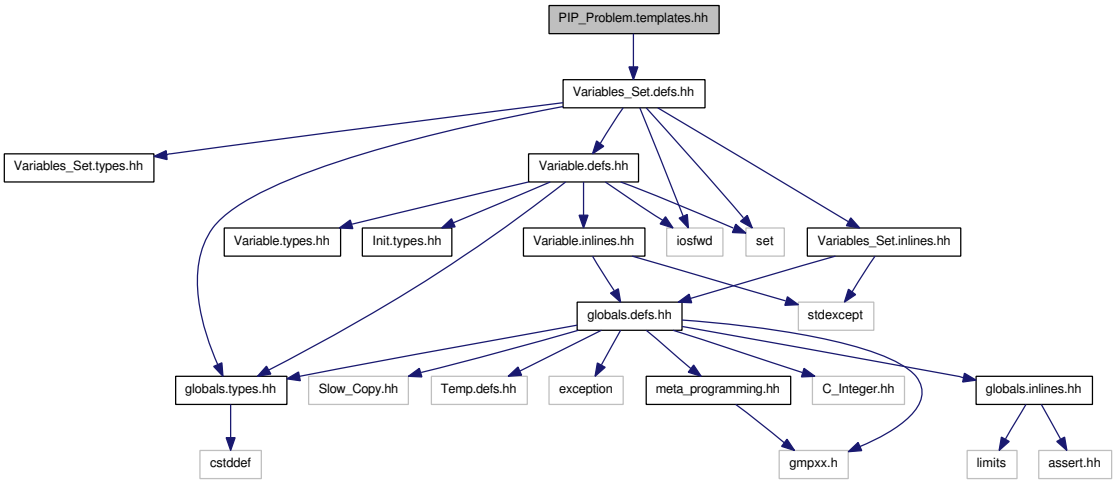
- namespace std

  The standard C++ namespace.

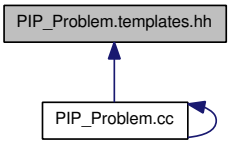
12.185 PIP_Problem.templates.hh File Reference

```
#include "Variables_Set.defs.hh"
```

Include dependency graph for PIP_Problem.templates.hh:



This graph shows which files directly or indirectly include this file:



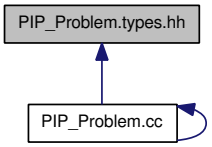
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.186 PIP_Problem.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Enumerations

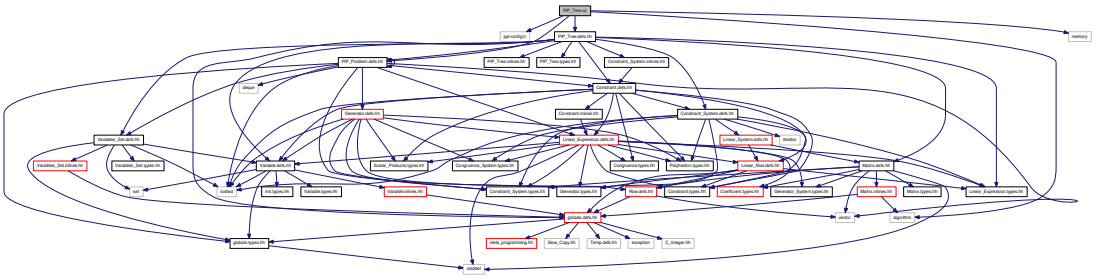
- enum Parma_Polyhedra_Library::PIP_Problem_Status { Parma_Polyhedra_Library::UNFEASIBLE_PIP_PROBLEM, Parma_Polyhedra_Library::OPTIMIZED_PIP_PROBLEM }

  Possible outcomes of the PIP_Problem solver.

12.187 PIP_Tree.cc File Reference

```
#include <ppl-config.h>
#include "PIP_Tree.defs.hh"
#include "PIP_Problem.defs.hh"
#include <algorithm>
#include <memory>
```

Include dependency graph for PIP_Tree.cc:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

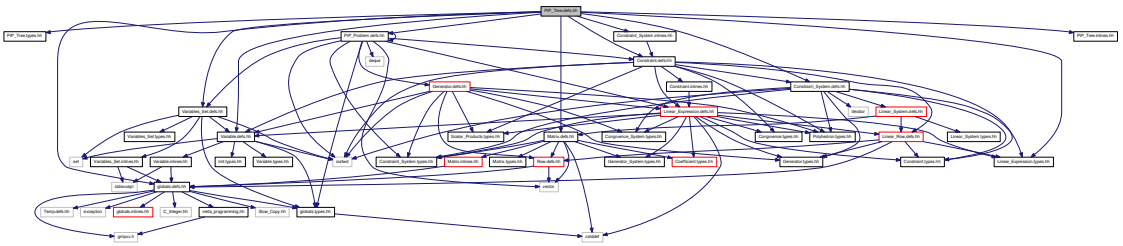
- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

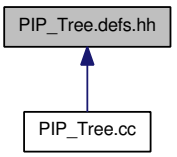
12.188 PIP_Tree.defs.hh File Reference

```
#include "PIP_Tree.types.hh"
#include "Variable.defs.hh"
#include "Linear_Expression.types.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Constraint.defs.hh"
#include "Matrix.defs.hh"
#include "Variables_Set.defs.hh"
#include "globals.defs.hh"
#include "PIP_Problem.defs.hh"
#include "PIP_Tree.inlines.hh"
```

Include dependency graph for PIP_Tree.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::PIP_Tree_Node

  A node of the PIP solution tree.

- class Parma_Polyhedra_Library::PIP_Tree_Node::Artificial_Parameter

  Artificial parameters in PIP solution trees.

- class Parma_Polyhedra_Library::PIP_Solution_Node

  A tree node representing part of the space of solutions.

- struct Parma_Polyhedra_Library::PIP_Solution_Node::Tableau

  The type for parametric simplex tableau.

- struct Parma_Polyhedra_Library::PIP_Solution_Node::No_Constraints

  A tag type to select the alternative copy constructor.

- class Parma_Polyhedra_Library::PIP_Decision_Node

  A tree node representing a decision in the space of solutions.

Namespaces

- namespace Parma_Polyhedra_Library

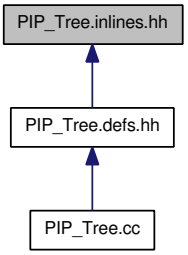
  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

12.189 PIP_Tree.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



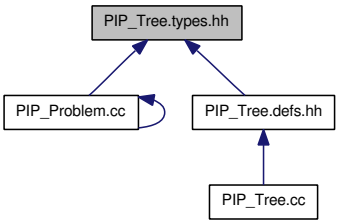
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.190 PIP_Tree.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Typedefs

- typedef const PIP_Tree_Node ∗ Parma_Polyhedra_Library::PIP_Tree

12.191 Pointset_Powerset.cc File Reference

```
#include <ppl-config.h>
#include "Pointset_Powerset.defs.hh"
#include "Grid.defs.hh"
```

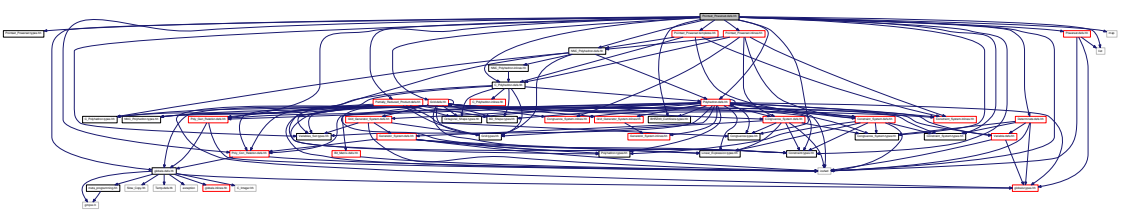
```
#include <utility>
```

Include dependency graph for Pointset_Powerset.cc:

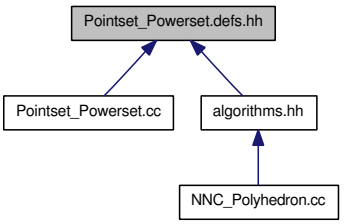
12.192 Pointset_Powerset.defs.hh File Reference

```
#include "Pointset_Powerset.types.hh"
#include "globals.defs.hh"
#include "BHRZ03_Certificate.types.hh"
#include "Constraint.types.hh"
#include "Constraint_System.types.hh"
#include "Congruence.types.hh"
#include "Congruence_System.types.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include "Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "Partially_Reduced_Product.defs.hh"
#include "Variables_Set.types.hh"
#include "Determinate.defs.hh"
#include "Powerset.defs.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include <iosfwd>
#include <list>
#include <map>
#include "Pointset_Powerset.inlines.hh"
#include "Pointset_Powerset.templates.hh"
```

Include dependency graph for Pointset_Powerset.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Pointset_Powerset< PSET >

  The powerset construction instantiated on PPL pointset domains.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

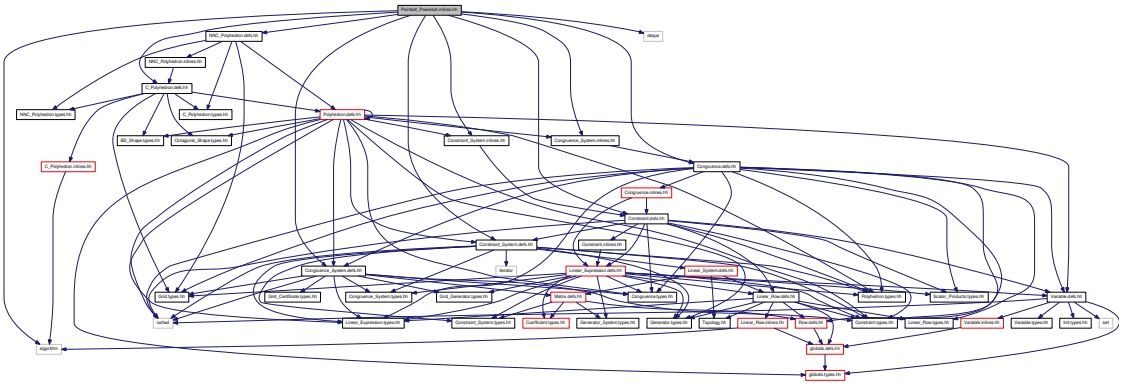
- namespace std

  The standard C++ namespace.

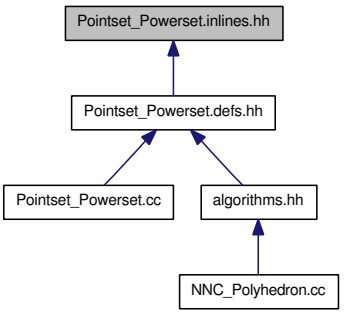
12.193 Pointset_Powerset.inlines.hh File Reference

```
#include "Constraint.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence.defs.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include <algorithm>
#include <deque>
```

Include dependency graph for Pointset_Powerset.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

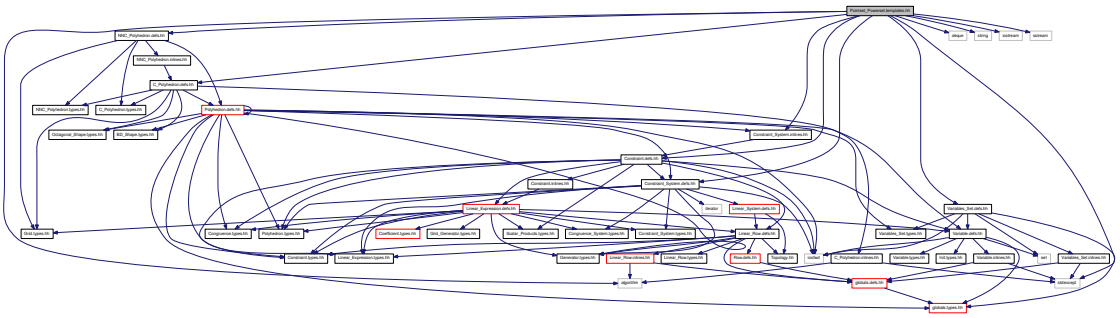
  The standard C++ namespace.

12.194 Pointset_Powerset.templates.hh File Reference

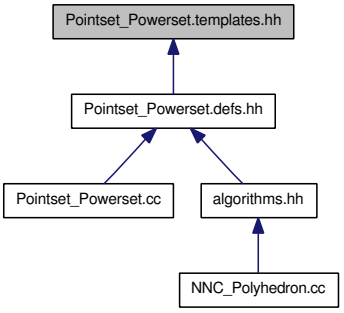
```
#include "Constraint.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include "Variables_Set.defs.hh"
#include <algorithm>
```

```
#include <deque>
#include <string>
#include <iostream>
#include <sstream>
#include <stdexcept>
```

Include dependency graph for Pointset_Powerset.templates.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

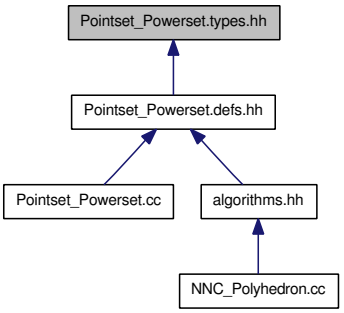
- namespace Parma_Polyhedra_Library::Implementation

  Implementation related data and functions.

- namespace Parma_Polyhedra_Library::Implementation::Pointset_Powersets

12.195 Pointset_Powerset.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

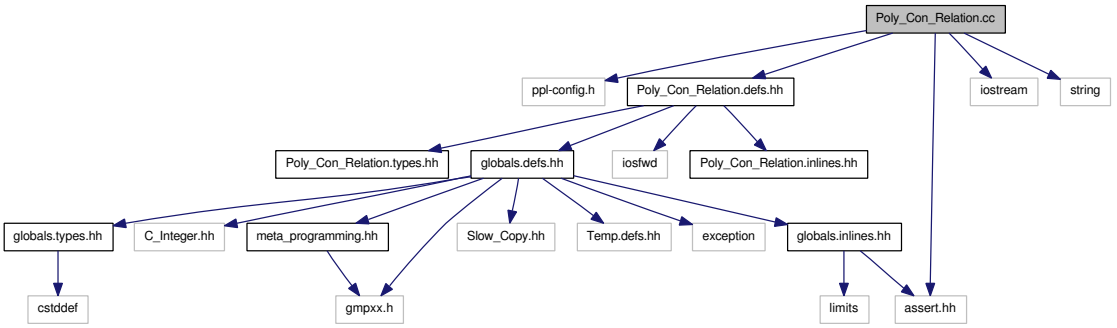
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.196 Poly_Con_Relation.cc File Reference

```
#include <ppl-config.h>
#include "Poly_Con_Relation.defs.hh"
#include <iostream>
#include <string>
#include "assert.hh"
```

Include dependency graph for Poly_Con_Relation.cc:

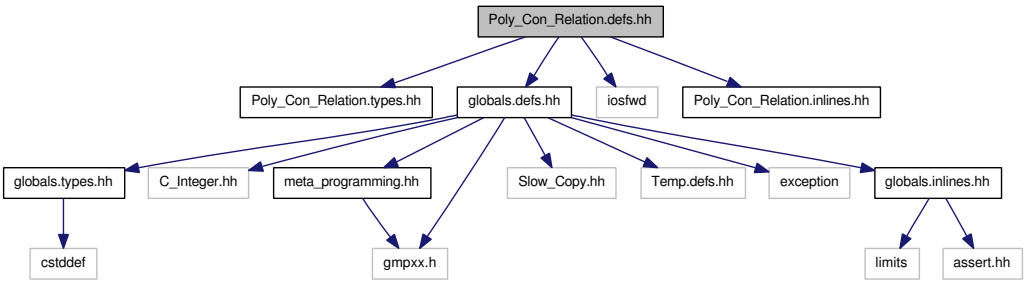


12.197 Poly_Con_Relation.defs.hh File Reference

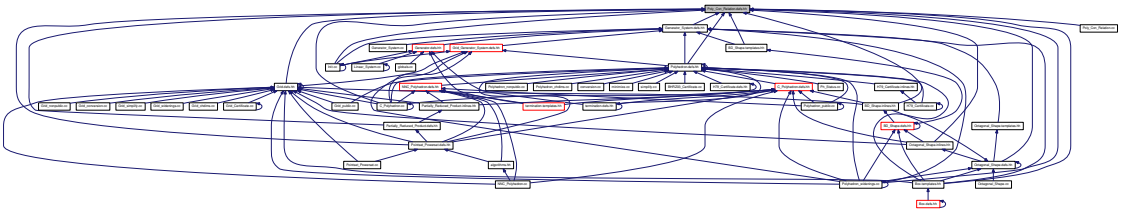
```
#include "Poly_Con_Relation.types.hh"
#include "globals.defs.hh"
#include <iosfwd>
```

```
#include "Poly_Con_Relation.inlines.hh"
```

Include dependency graph for Poly_Con_Relation.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Poly_Con_Relation

  The relation between a polyhedron and a constraint.

Namespaces

- namespace Parma_Polyhedra_Library

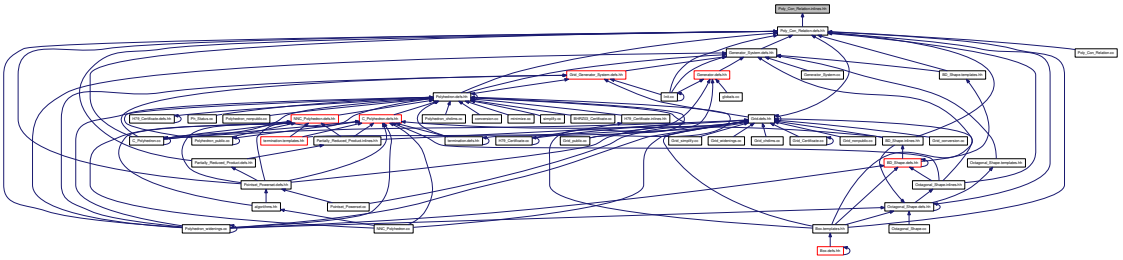
  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

12.198 Poly_Con_Relation.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



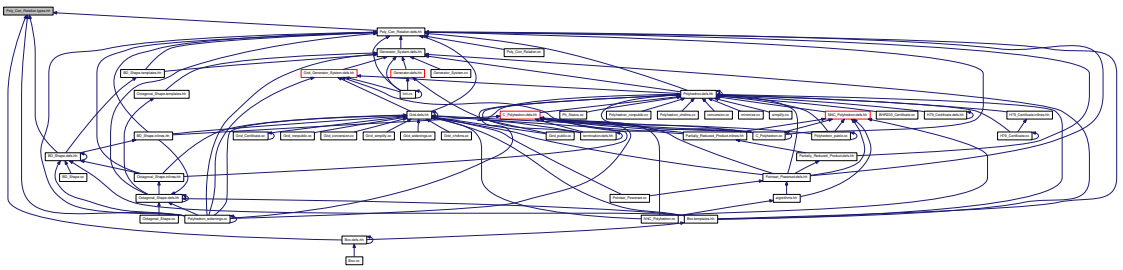
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.199 Poly_Con_Relation.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

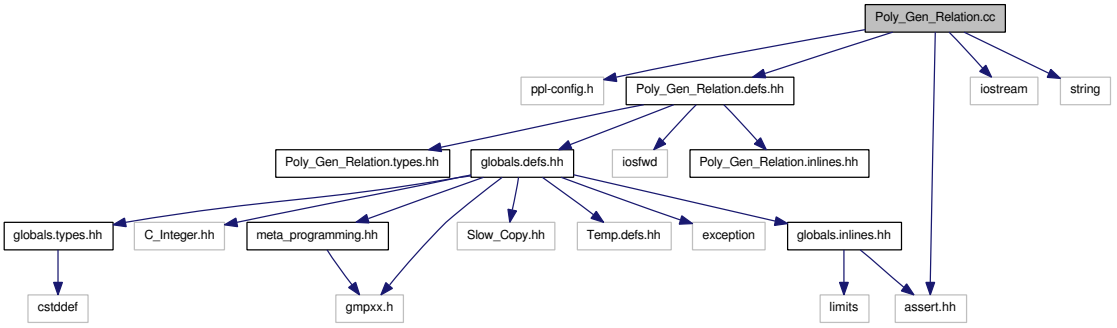
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.200 Poly_Gen_Relation.cc File Reference

```
#include <ppl-config.h>
#include "Poly_Gen_Relation.defs.hh"
#include <iostream>
#include <string>
#include "assert.hh"
```

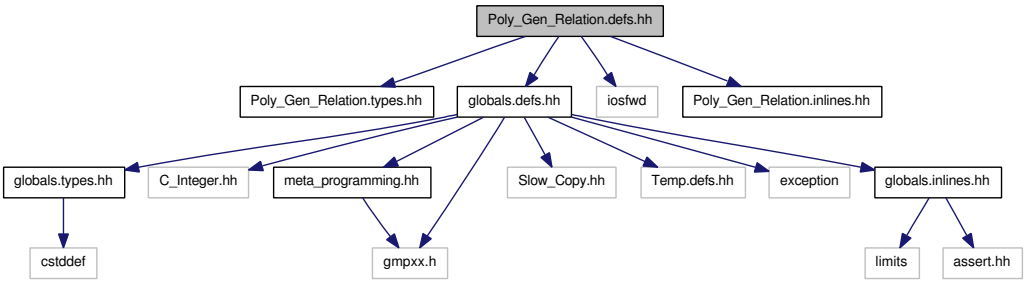
Include dependency graph for Poly_Gen_Relation.cc:



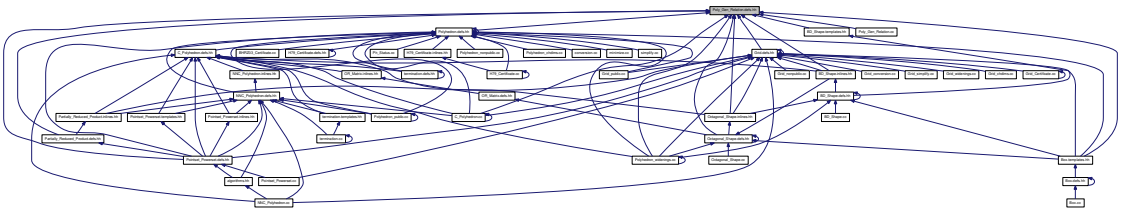
12.201 Poly_Gen_Relation.defs.hh File Reference

```
#include "Poly_Gen_Relation.types.hh"
#include "globals.defs.hh"
#include <iosfwd>
#include "Poly_Gen_Relation.inlines.hh"
```

Include dependency graph for Poly_Gen_Relation.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Poly_Gen_Relation

  The relation between a polyhedron and a generator.

Namespaces

- namespace Parma_Polyhedra_Library

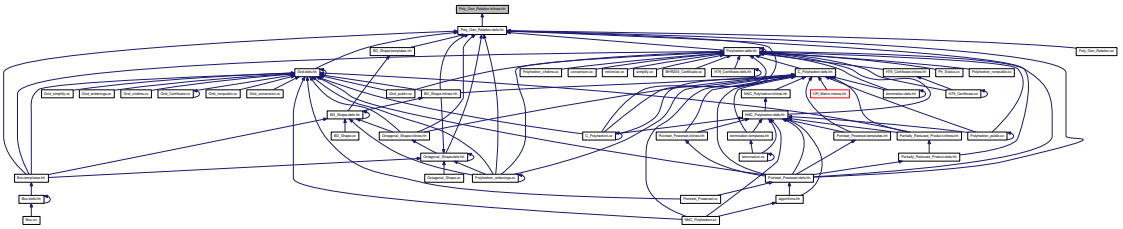
  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

12.202 Poly_Gen_Relation.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



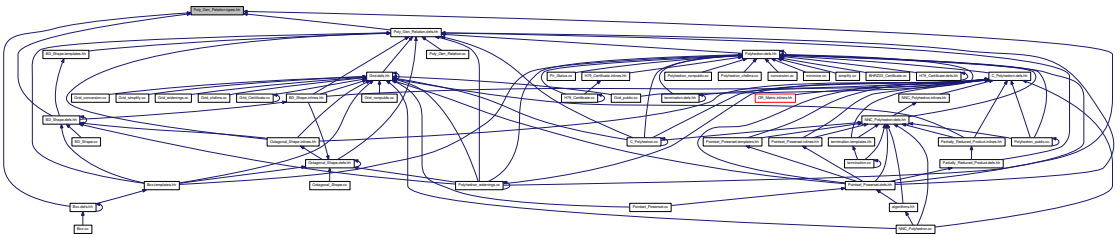
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.203 Poly_Gen_Relation.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

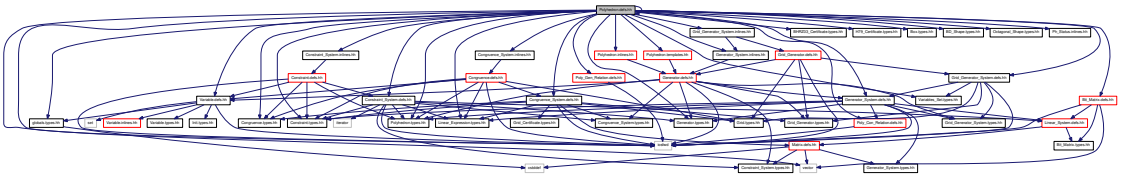
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

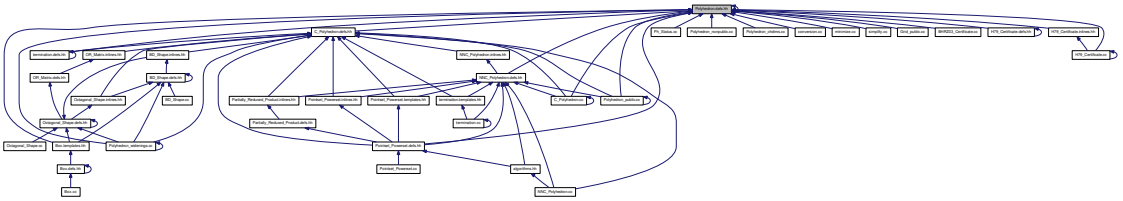
12.204 Polyhedron.defs.hh File Reference

```
#include "Polyhedron.types.hh"
#include "globals.types.hh"
#include "Variable.defs.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include "Bit_Matrix.defs.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "BHRZ03_Certificate.types.hh"
#include "H79_Certificate.types.hh"
#include "Box.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include <vector>
#include <iosfwd>
#include "Ph_Status.idefs.hh"
#include "Ph_Status.inlines.hh"
#include "Polyhedron.inlines.hh"
#include "Polyhedron.templates.hh"
```

Include dependency graph for Polyhedron.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Polyhedron

  The base class for convex polyhedra.

- class Parma_Polyhedra_Library::Polyhedron::Status

  A conjunctive assertion about a polyhedron.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

- namespace Parma_Polyhedra_Library::Interfaces

  Data and functions related to language interfaces.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_IN_Polyhedron_CLASS

Functions

- bool Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces (const Polyhedron &ph)

  Returns `true` *if and only if* `ph.topology() == NECESSARILY_CLOSED`.

12.204.1 Define Documentation

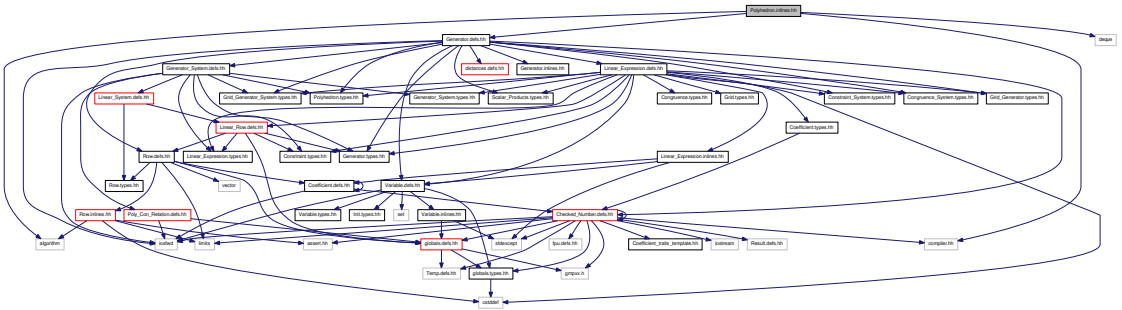
12.204.1.1 #define PPL_IN_Polyhedron_CLASS

Definition at line 1928 of file Polyhedron.defs.hh.

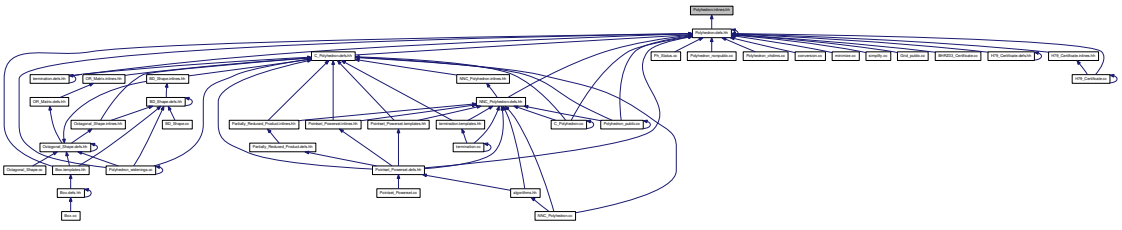
12.205 Polyhedron.inlines.hh File Reference

```
#include "Generator.defs.hh"
#include "compiler.hh"
#include <algorithm>
#include <deque>
```

Include dependency graph for Polyhedron.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Interfaces
  Data and functions related to language interfaces.
- namespace std
  The standard C++ namespace.

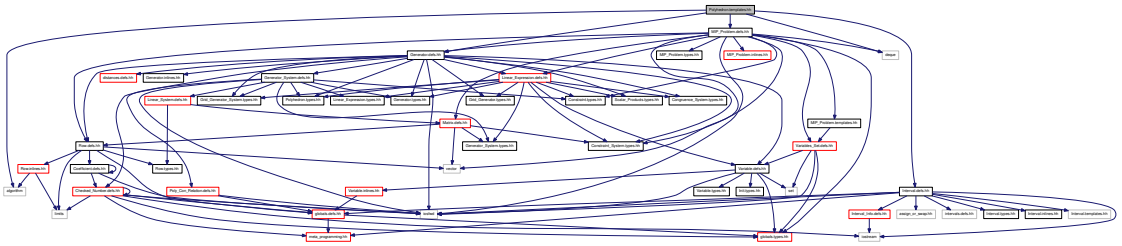
Functions

- bool Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces (const Polyhedron &ph)
  Returns `true` *if and only if* `ph.topology() == NECESSARILY_CLOSED`.

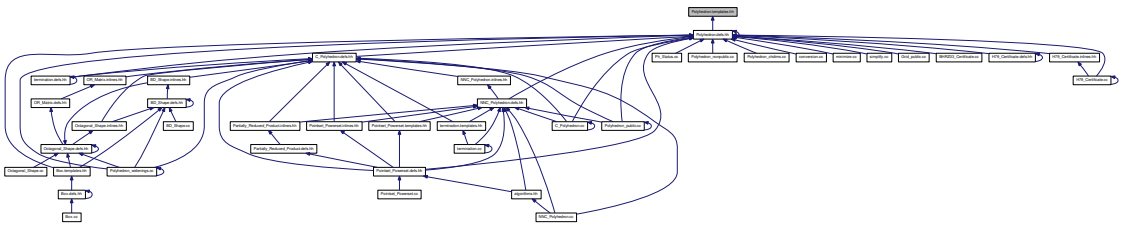
12.206 Polyhedron.templates.hh File Reference

```
#include "Generator.defs.hh"
#include "MIP_Problem.defs.hh"
#include "Interval.defs.hh"
#include <algorithm>
#include <deque>
```

Include dependency graph for Polyhedron.templates.hh:



This graph shows which files directly or indirectly include this file:

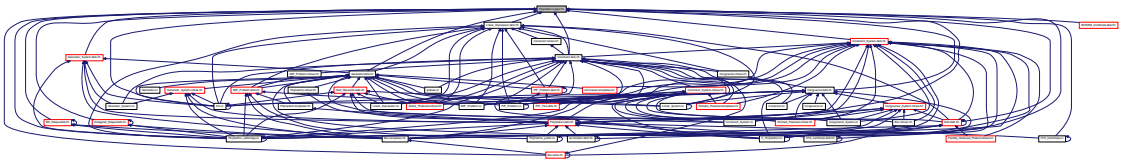


Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

12.207 Polyhedron.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

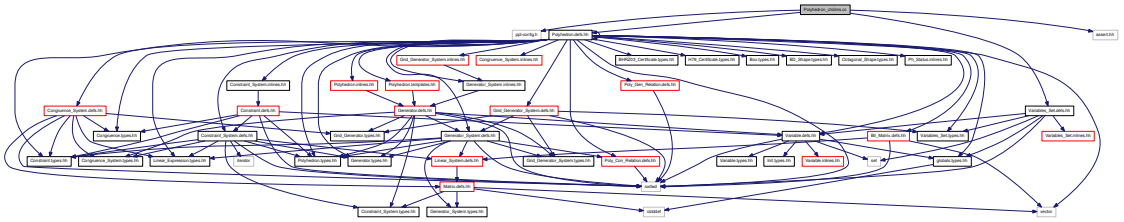
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.208 Polyhedron_chdims.cc File Reference

```
#include <ppl-config.h>
#include "Polyhedron.defs.hh"
#include "Variables_Set.defs.hh"
#include "assert.hh"
```

Include dependency graph for Polyhedron_chdims.cc:



Defines

- #define BE_LAZY 1

12.208.1 Define Documentation

12.208.1.1 #define BE_LAZY 1

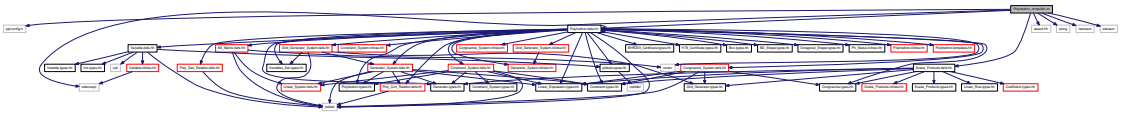
Definition at line 31 of file Polyhedron_chdims.cc.

12.209 Polyhedron_nonpublic.cc File Reference

```
#include <ppl-config.h>
```

```
#include "Polyhedron.defs.hh"
#include "Scalar_Products.defs.hh"
#include "assert.hh"
#include <string>
#include <iostream>
#include <sstream>
#include <stdexcept>
```

Include dependency graph for Polyhedron_nonpublic.cc:



Defines

- #define BE_LAZY 1

  Controls the laziness level of the implementation.

12.209.1 Define Documentation

12.209.1.1 #define BE_LAZY 1

Controls the laziness level of the implementation.

Temporarily used in a few of the function implementations to switch to an even more lazy algorithm. To be removed as soon as we collect enough information to decide which is the better implementation alternative.

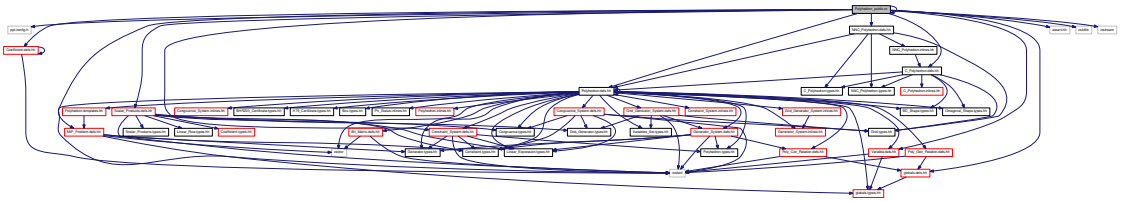
Definition at line 46 of file Polyhedron_nonpublic.cc.

12.210 Polyhedron_public.cc File Reference

```
#include <ppl-config.h>
#include "Polyhedron.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include "Scalar_Products.defs.hh"
#include "MIP_Problem.defs.hh"
#include "wrap_assign.hh"
#include "globals.defs.hh"
#include "Coefficient.defs.hh"
#include "Variable.defs.hh"
```

```
#include "Constraint_System.defs.hh"
#include "assert.hh"
#include <cstdlib>
#include <iostream>
```

Include dependency graph for Polyhedron_public.cc:



This graph shows which files directly or indirectly include this file:

Defines

- #define ENSURE_SORTEDNESS 0

12.210.1 Define Documentation

12.210.1.1 #define ENSURE_SORTEDNESS 0

Definition at line 36 of file Polyhedron_public.cc.

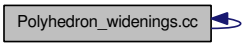
12.211 Polyhedron_widenings.cc File Reference

```
#include <ppl-config.h>
#include "Polyhedron.defs.hh"
#include "BHRZ03_Certificate.defs.hh"
#include "BHRZ03_Certificate.types.hh"
#include "Polyhedron.types.hh"
#include "globals.types.hh"
#include "assert.hh"
#include <vector>
#include "Interval.defs.hh"
#include <gmpxx.h>
#include "Box.types.hh"
```

```
#include "Coefficient.defs.hh"
#include "Variable.types.hh"
#include "Variables_Set.types.hh"
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Constraint.defs.hh"
#include "Constraint_System.types.hh"
#include "Generator.types.hh"
#include "Generator_System.types.hh"
#include "Congruence.types.hh"
#include "Congruence_System.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include "Poly_Con_Relation.types.hh"
#include "Poly_Gen_Relation.types.hh"
#include "Grid.types.hh"
#include "intervals.defs.hh"
#include <iosfwd>
#include <string>
#include "Boundary.defs.hh"
#include "Constraint_System.defs.hh"
#include "Constraint_System.inlines.hh"
#include "Congruence_System.defs.hh"
#include "Congruence_System.inlines.hh"
#include "distances.defs.hh"
#include "Variables_Set.defs.hh"
#include "Generator_System.defs.hh"
#include "Generator_System.inlines.hh"
#include "Poly_Con_Relation.defs.hh"
#include "Poly_Gen_Relation.defs.hh"
#include "globals.defs.hh"
#include "Variable.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Grid_Generator_System.defs.hh"
#include "Grid_Generator_System.inlines.hh"
#include "Grid_Generator.types.hh"
```

```
#include "Grid_Certificate.types.hh"
#include "Grid_Generator.defs.hh"
#include <algorithm>
#include <deque>
#include <iterator>
#include <cstddef>
#include "checked.defs.hh"
#include <limits>
#include <iostream>
#include "Checked_Number.types.hh"
#include "Rounding_Dir.defs.hh"
#include "Checked_Number.defs.hh"
#include "DB_Row.defs.hh"
#include "meta_programming.hh"
#include "Bit_Matrix.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "Grid.defs.hh"
#include "WRD_coefficient_types.defs.hh"
#include "Bit_Row.defs.hh"
#include <climits>
#include "BD_Shape.defs.hh"
#include "wrap_assign.hh"
#include <sstream>
#include <stdexcept>
#include "Temp.defs.hh"
#include "MIP_Problem.defs.hh"
#include "Octagonal_Shape.defs.hh"
#include "Rational_Interval.hh"
#include <map>
#include "Scalar_Products.defs.hh"
```

This graph shows which files directly or indirectly include this file:

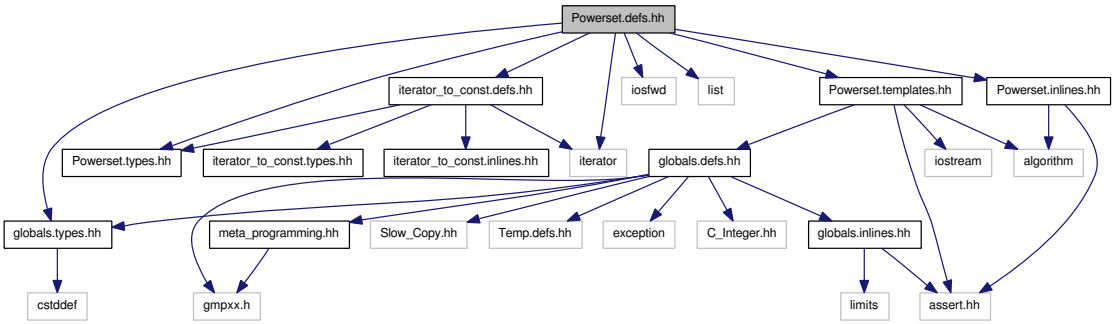


12.212 Powerset.defs.hh File Reference

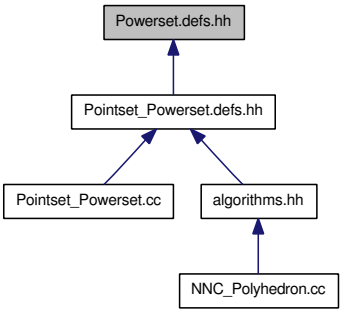
```
#include "Powerset.types.hh"
```

```
#include "globals.types.hh"
#include "iterator_to_const.defs.hh"
#include <iosfwd>
#include <iterator>
#include <list>
#include "Powerset.inlines.hh"
#include "Powerset.templates.hh"
```

Include dependency graph for Powerset.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Powerset< D >

  The powerset construction on a base-level domain.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

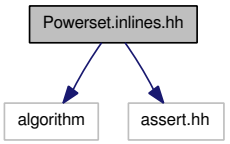
- namespace std

  The standard C++ namespace.

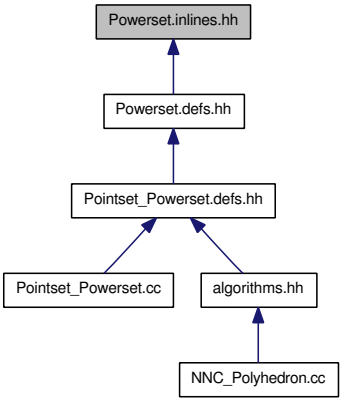
12.213 Powerset.inlines.hh File Reference

```
#include <algorithm>
#include "assert.hh"
```

Include dependency graph for Powerset.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

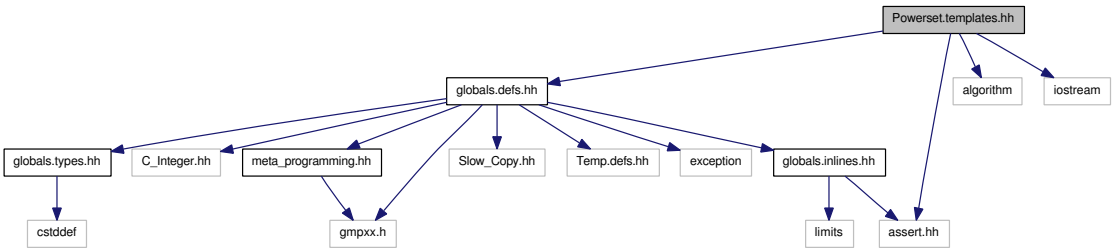
  The standard C++ namespace.

12.214 Powerset.templates.hh File Reference

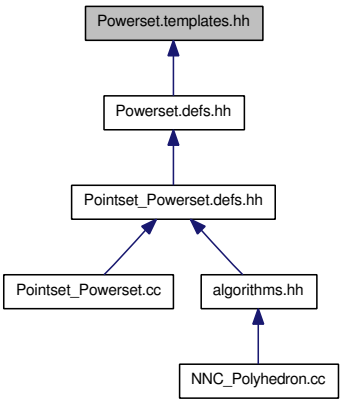
```
#include "globals.defs.hh"
#include <algorithm>
#include "assert.hh"
```

```
#include <iostream>
```

Include dependency graph for Powerset.templates.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

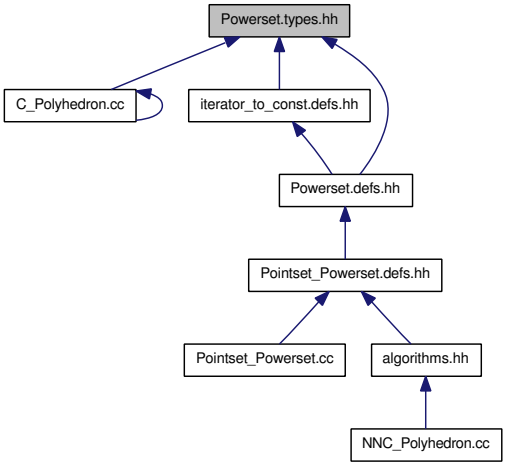
  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

12.215 Powerset.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

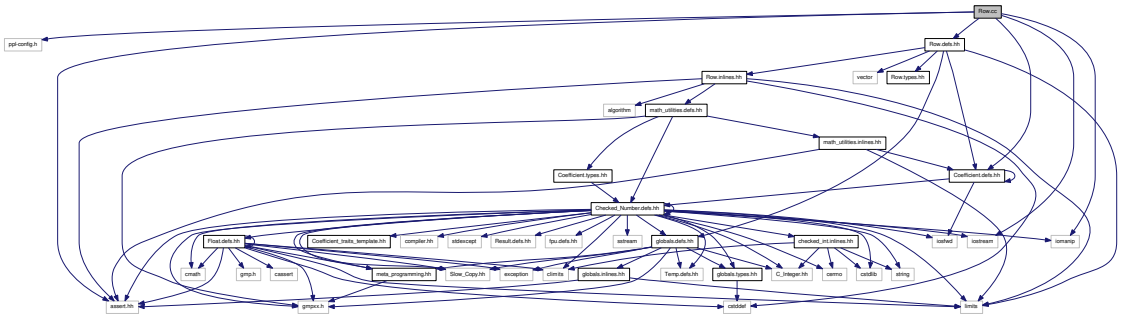
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.216 Row.cc File Reference

```
#include <ppl-config.h>
#include "Row.defs.hh"
#include "Coefficient.defs.hh"
#include <iostream>
#include <iomanip>
#include "assert.hh"
```

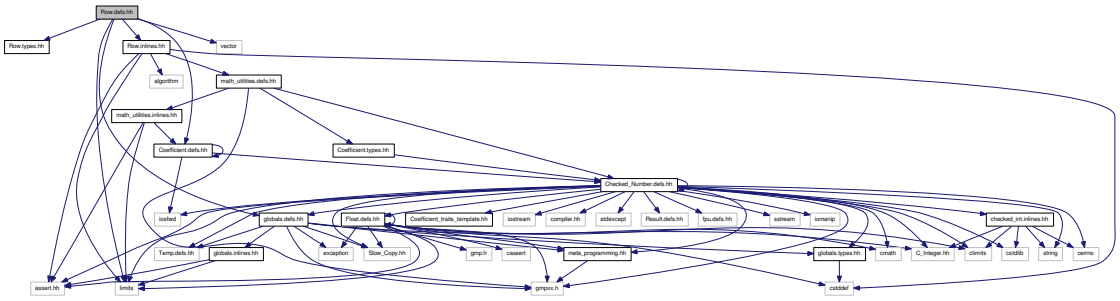
Include dependency graph for Row.cc:



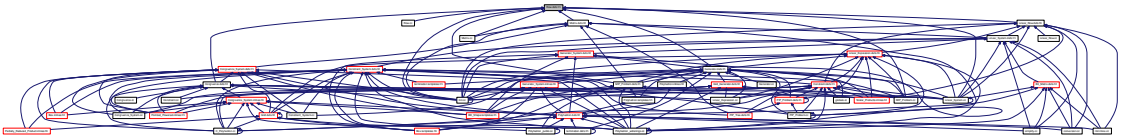
12.217 Row.defs.hh File Reference

```
#include "Row.types.hh"
#include "globals.defs.hh"
#include "Coefficient.defs.hh"
#include <vector>
#include <limits>
#include "Row.inlines.hh"
```

Include dependency graph for Row.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Row_Impl_Handler

  The handler of the actual Row implementation.

- class Parma_Polyhedra_Library::Row

  A finite sequence of coefficients.

- class Parma_Polyhedra_Library::Row::Flags

  Wrapper class to represent a set of flags with bits in a native unsigned integral type.

- class Parma_Polyhedra_Library::Row_Impl_Handler::Impl

  The actual implementation of a Row object.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

Defines

- #define PPL_ROW_EXTRA_DEBUG 0

12.217.1 Define Documentation

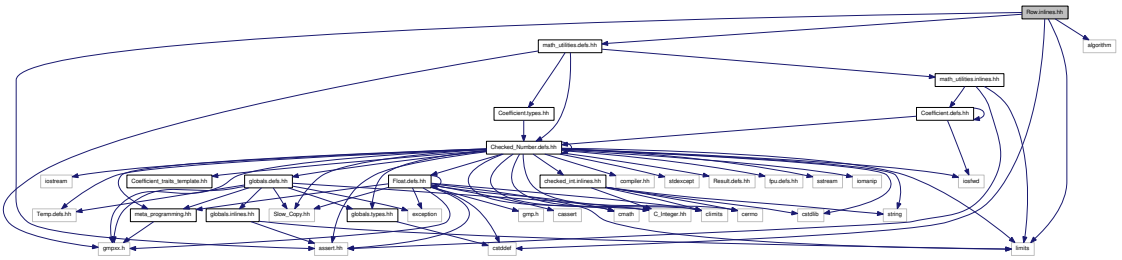
12.217.1.1 #define PPL_ROW_EXTRA_DEBUG 0

Definition at line 48 of file Row.defs.hh.

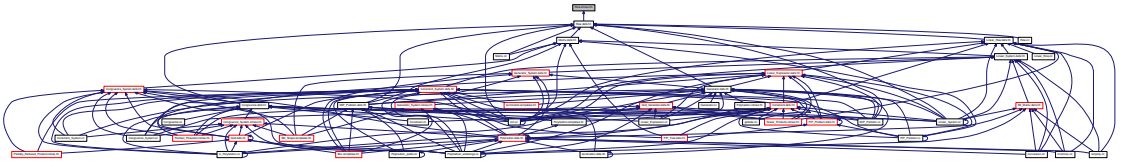
12.218 Row.inlines.hh File Reference

```
#include "math_utilities.defs.hh"
#include "assert.hh"
#include <cstddef>
#include <limits>
#include <algorithm>
```

Include dependency graph for Row.inlines.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

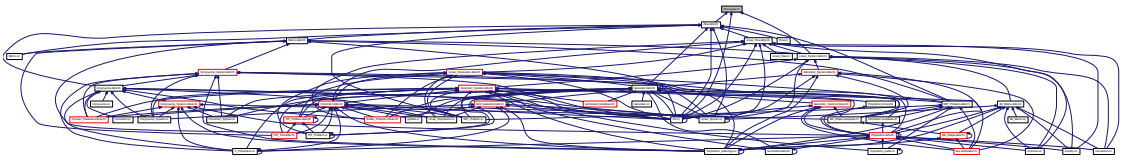
The entire library is confined to this namespace.

- namespace std

  The standard C++ namespace.

12.219 Row.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

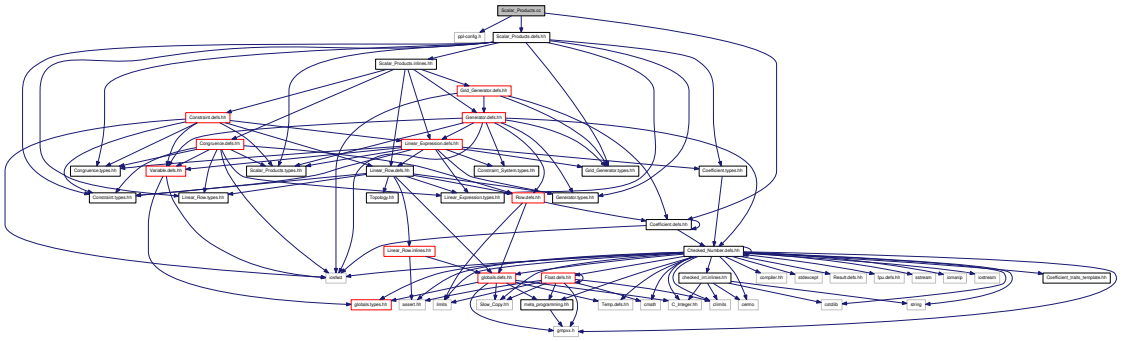
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.220 Scalar_Products.cc File Reference

```
#include <ppl-config.h>
#include "Scalar_Products.defs.hh"
#include "Coefficient.defs.hh"
```

Include dependency graph for Scalar_Products.cc:

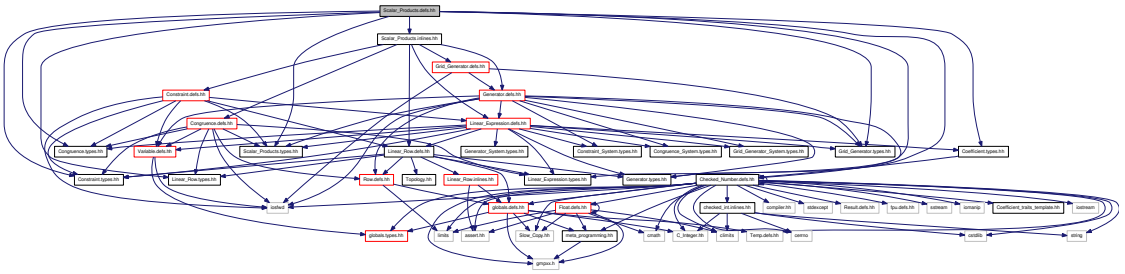


12.221 Scalar_Products.defs.hh File Reference

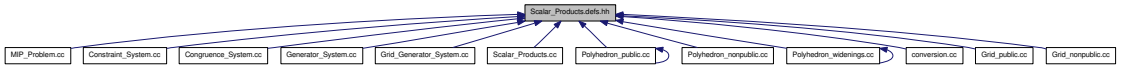
```
#include "Scalar_Products.types.hh"
#include "Linear_Row.types.hh"
#include "Coefficient.types.hh"
```

```
#include "Linear_Expression.types.hh"
#include "Constraint.types.hh"
#include "Generator.types.hh"
#include "Congruence.types.hh"
#include "Grid_Generator.types.hh"
#include "Scalar_Products.inlines.hh"
```

Include dependency graph for Scalar_Products.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Scalar_Products

  A class implementing various scalar product functions.

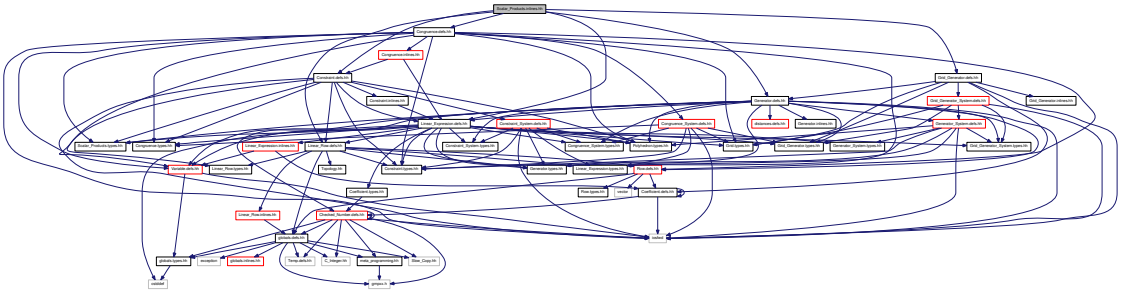
- class Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign

  Scalar product sign function object depending on topology.

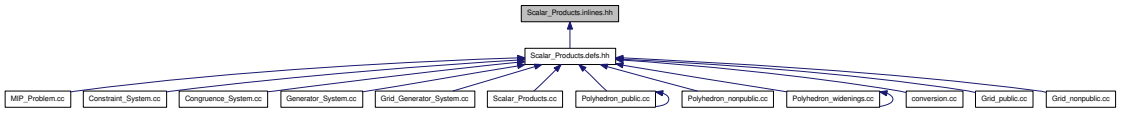
12.222 Scalar_Products.inlines.hh File Reference

```
#include "Linear_Row.defs.hh"
#include "Linear_Expression.defs.hh"
#include "Constraint.defs.hh"
#include "Generator.defs.hh"
#include "Congruence.defs.hh"
#include "Grid_Generator.defs.hh"
```

Include dependency graph for Scalar_Products.inlines.hh:



This graph shows which files directly or indirectly include this file:



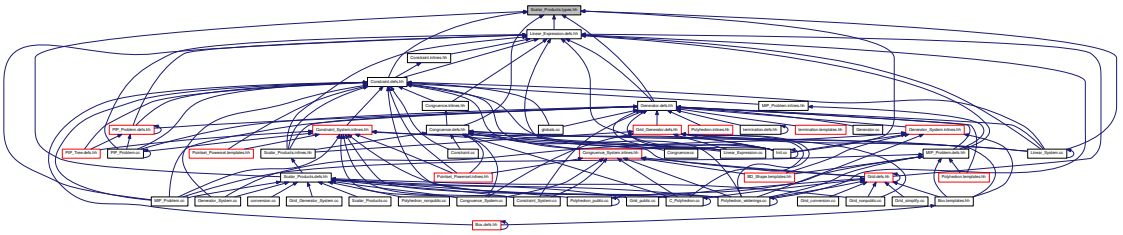
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.223 Scalar_Products.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

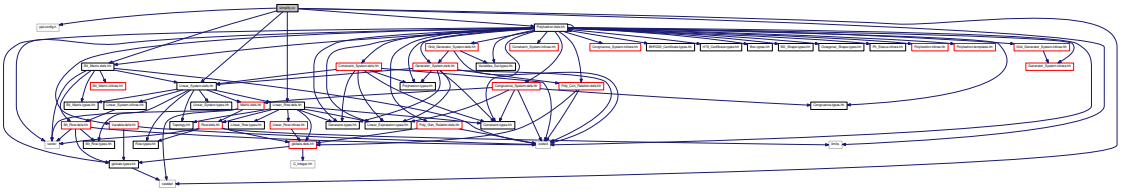
  The entire library is confined to this namespace.

12.224 simplify.cc File Reference

```
#include <ppl-config.h>
#include "Linear_Row.defs.hh"
```

```
#include "Linear_System.defs.hh"
#include "Bit_Matrix.defs.hh"
#include "Polyhedron.defs.hh"
#include <cstddef>
#include <limits>
```

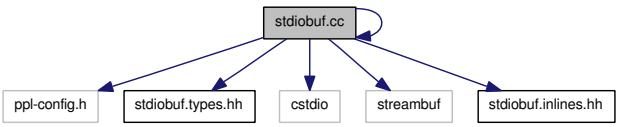
Include dependency graph for simplify.cc:



12.225 stdiobuf.cc File Reference

```
#include <ppl-config.h>
#include "stdiobuf.defs.hh"
#include "stdiobuf.types.hh"
#include <cstdio>
#include <streambuf>
#include "stdiobuf.inlines.hh"
```

Include dependency graph for stdiobuf.cc:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace Parma_Polyhedra_Library

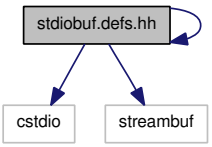
  The entire library is confined to this namespace.

12.226 stdiobuf.defs.hh File Reference

```
#include "stdiobuf.types.hh"
```

```
#include <cstdio>
#include <streambuf>
```

Include dependency graph for stdiobuf.defs.hh:



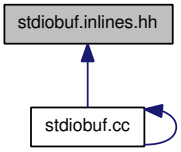
This graph shows which files directly or indirectly include this file:

Classes

- class Parma_Polyhedra_Library::stdiobuf

12.227 stdiobuf.inlines.hh File Reference

This graph shows which files directly or indirectly include this file:



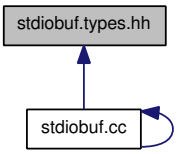
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.228 stdiobuf.types.hh File Reference

This graph shows which files directly or indirectly include this file:



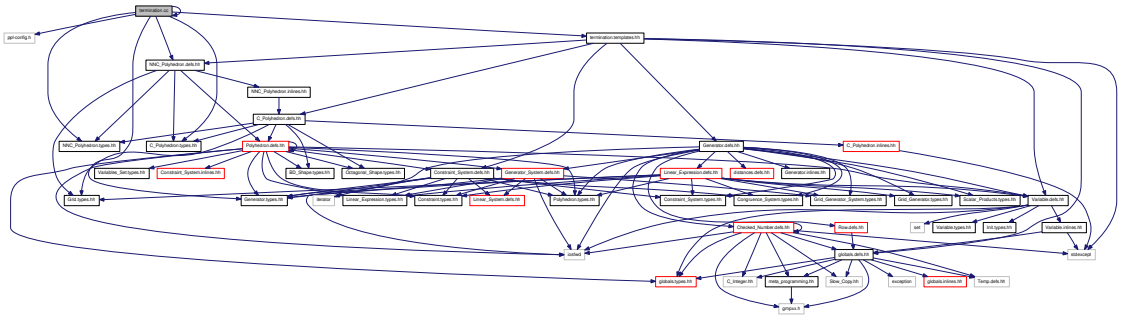
Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.

12.229 termination.cc File Reference

```
#include <ppl-config.h>
#include "termination.defs.hh"
#include "Generator.types.hh"
#include "C_Polyhedron.types.hh"
#include "NNC_Polyhedron.types.hh"
#include "termination.templates.hh"
#include "NNC_Polyhedron.defs.hh"
```

Include dependency graph for termination.cc:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace Parma_Polyhedra_Library
  The entire library is confined to this namespace.
- namespace Parma_Polyhedra_Library::Implementation
  Implementation related data and functions.
- namespace Parma_Polyhedra_Library::Implementation::Termination

Functions

- void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_-approximation (const Constraint_System &cs_in, Constraint_System &cs_out)

- template<>
  void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_-approximation (const C_Polyhedron &ph, Constraint_System &cs)
- void Parma_Polyhedra_Library::Implementation::Termination::shift_unprimed_variables (Constraint_System &cs)
- void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_systems_MS (const Constraint_System &cs, Constraint_System &cs_out1, Constraint_System &cs_out2)

  Fill the constraint system(s) for the application of the Mesnard and Serebrenik improved termination tests.

- void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_system_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, Constraint_System &cs_out, Linear_Expression &le_out)

  Fill the constraint system(s) for the application of the Podelski and Rybalchenko improved termination tests.

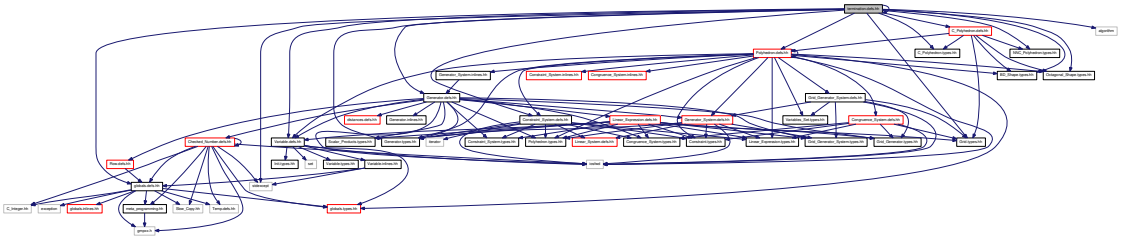
- void Parma_Polyhedra_Library::Implementation::Termination::fill_constraint_system_PR_original (const Constraint_System &cs, Constraint_System &cs_out, Linear_Expression &le_out)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_MS (const Constraint_System &cs)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_MS (const Constraint_System &cs, Generator &mu)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_MS (const Constraint_System &cs, C_Polyhedron &mu_space)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_quasi_ranking_-functions_MS (const Constraint_System &cs, C_Polyhedron &decreasing_mu_space, C_-Polyhedron &bounded_mu_space)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR_original (const Constraint_System &cs)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR (const Constraint_System &cs_before, const Constraint_System &cs_after)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, Generator &mu)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR_-original (const Constraint_System &cs, Generator &mu)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, NNC_Polyhedron &mu_space)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR_-original (const Constraint_System &cs, NNC_Polyhedron &mu_space)

12.230 termination.defs.hh File Reference

```
#include "Generator.types.hh"
#include "C_Polyhedron.types.hh"
#include "globals.defs.hh"
#include "Variable.defs.hh"
#include "Generator.defs.hh"
#include "Constraint_System.defs.hh"
#include "C_Polyhedron.types.hh"
```

```
#include "NNC_Polyhedron.types.hh"
#include "Polyhedron.defs.hh"
#include "Grid.types.hh"
#include "BD_Shape.types.hh"
#include "Octagonal_Shape.types.hh"
#include <algorithm>
#include <stdexcept>
#include "C_Polyhedron.defs.hh"
```

Include dependency graph for termination.defs.hh:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Functions

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_MS (const PSET &pset)

  Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_MS_2 (const PSET &pset_before, const PSET &pset_after)

  Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_MS (const PSET &pset, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_MS_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_MS (const PSET &pset, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS (const PSET &pset, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_PR (const PSET &pset)

  Like termination_test_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_PR_2 (const PSET &pset_before, const PSET &pset_after)

  Like termination_test_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_PR (const PSET &pset, Generator &mu)

  Like one_affine_ranking_function_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_PR_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

  Like one_affine_ranking_function_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_PR (const PSET &pset, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

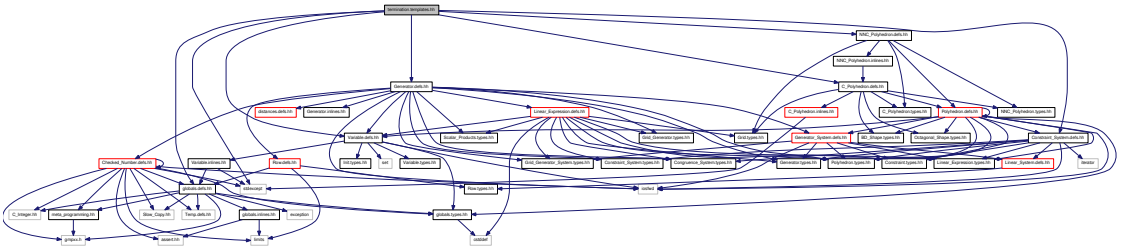
- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_PR_2 (const PSET &pset_before, const PSET &pset_after, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

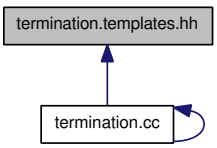
12.231 termination.templates.hh File Reference

```
#include "globals.defs.hh"
#include "Variable.defs.hh"
#include "Generator.defs.hh"
#include "Constraint_System.defs.hh"
#include "C_Polyhedron.defs.hh"
#include "NNC_Polyhedron.defs.hh"
#include <stdexcept>
```

Include dependency graph for termination.templates.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

- namespace Parma_Polyhedra_Library::Implementation

Implementation related data and functions.

- namespace Parma_Polyhedra_Library::Implementation::Termination

Defines

- #define PRINT_DEBUG_INFO 0

Functions

- void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const Constraint_System &cs_in, Constraint_System &cs_out)
- template<typename PSET >
  void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const PSET &pset, Constraint_System &cs)
- template<>
  void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const C_Polyhedron &ph, Constraint_System &cs)
- void Parma_Polyhedra_Library::Implementation::Termination::shift_unprimed_variables (Constraint_System &cs)
- template<typename PSET >
  void Parma_Polyhedra_Library::Implementation::Termination::assign_all_inequalities_approximation (const PSET &pset_before, const PSET &pset_after, Constraint_System &cs)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_MS (const Constraint_System &cs)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_MS (const Constraint_System &cs, Generator &mu)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_MS (const Constraint_System &cs, C_Polyhedron &mu_space)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_quasi_ranking_functions_MS (const Constraint_System &cs, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR (const Constraint_System &cs_before, const Constraint_System &cs_after)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, Generator &mu)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR (const Constraint_System &cs_before, const Constraint_System &cs_after, NNC_Polyhedron &mu_space)
- bool Parma_Polyhedra_Library::Implementation::Termination::termination_test_PR_original (const Constraint_System &cs)
- bool Parma_Polyhedra_Library::Implementation::Termination::one_affine_ranking_function_PR_original (const Constraint_System &cs, Generator &mu)
- void Parma_Polyhedra_Library::Implementation::Termination::all_affine_ranking_functions_PR_original (const Constraint_System &cs, NNC_Polyhedron &mu_space)
- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_MS (const PSET &pset)

  Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_MS_2 (const PSET &pset_before, const PSET &pset_after)

Termination test using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_MS (const PSET &pset, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_MS_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

  Termination test with witness ranking function using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_MS (const PSET &pset, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &mu_space)

  Termination test with ranking function space using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS (const PSET &pset, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_quasi_ranking_functions_MS_2 (const PSET &pset_before, const PSET &pset_after, C_Polyhedron &decreasing_mu_space, C_Polyhedron &bounded_mu_space)

  Computes the spaces of affine quasi ranking functions using an improvement of the method by Mesnard and Serebrenik [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_PR_2 (const PSET &pset_before, const PSET &pset_after)

  Like termination_test_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::termination_test_PR (const PSET &pset)

  Like termination_test_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_PR_2 (const PSET &pset_before, const PSET &pset_after, Generator &mu)

Like one_affine_ranking_function_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  bool Parma_Polyhedra_Library::one_affine_ranking_function_PR (const PSET &pset, Generator &mu)

  Like one_affine_ranking_function_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_PR_2 (const PSET &pset_before, const PSET &pset_after, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS_2() but using an alternative formalization of the method by Podelski and Rybalchenko [BMPZ10].

- template<typename PSET >
  void Parma_Polyhedra_Library::all_affine_ranking_functions_PR (const PSET &pset, NNC_Polyhedron &mu_space)

  Like all_affine_ranking_functions_MS() but using the method by Podelski and Rybalchenko [BMPZ10].

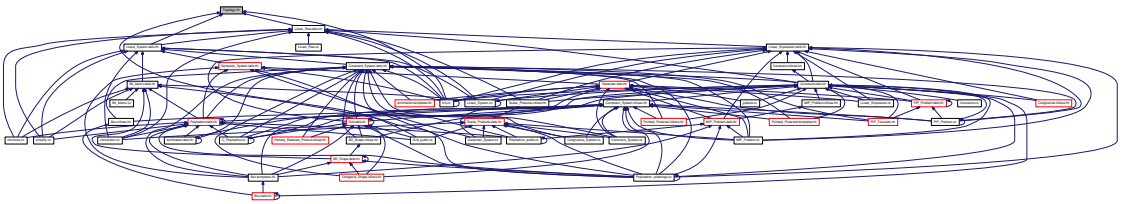
12.231.1 Define Documentation

12.231.1.1 #define PRINT_DEBUG_INFO 0

Definition at line 36 of file termination.templates.hh.

12.232 Topology.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Enumerations

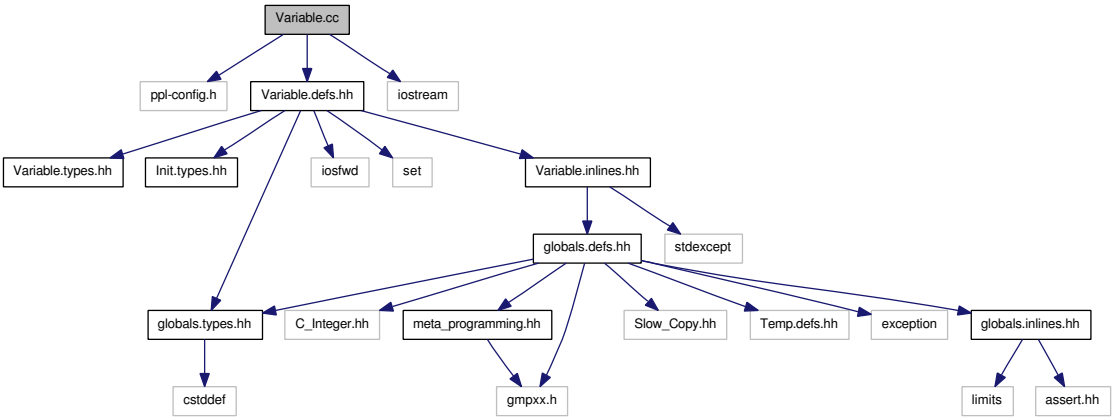
- enum Parma_Polyhedra_Library::Topology { Parma_Polyhedra_Library::NECESSARILY_CLOSED = 0, Parma_Polyhedra_Library::NOT_NECESSARILY_CLOSED = 1 }

Kinds of polyhedra domains.

12.233 Variable.cc File Reference

```
#include <ppl-config.h>
#include "Variable.defs.hh"
#include <iostream>
```

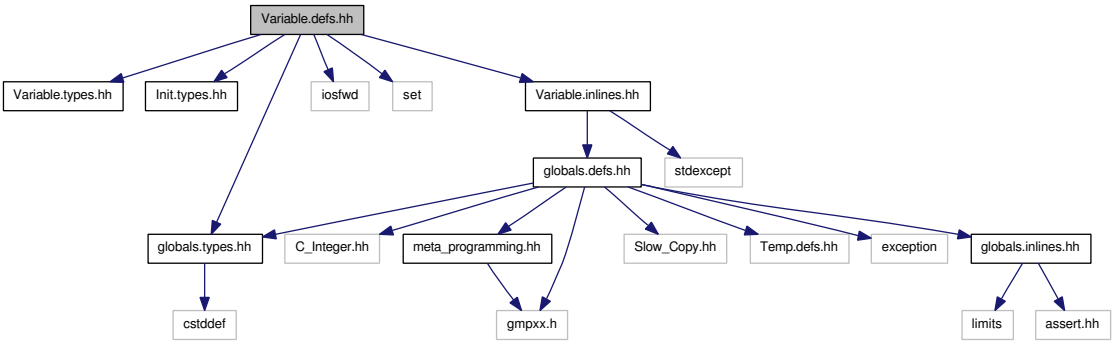
Include dependency graph for Variable.cc:



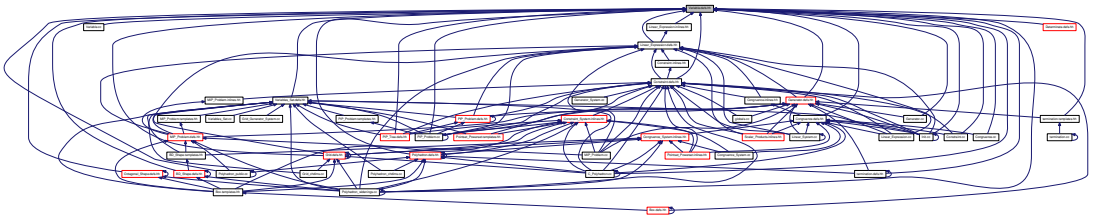
12.234 Variable.defs.hh File Reference

```
#include "Variable.types.hh"
#include "Init.types.hh"
#include "globals.types.hh"
#include <iosfwd>
#include <set>
#include "Variable.inlines.hh"
```

Include dependency graph for Variable.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Variable

  A dimension of the vector space.

- struct Parma_Polyhedra_Library::Variable::Compare

  Binary predicate defining the total ordering on variables.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

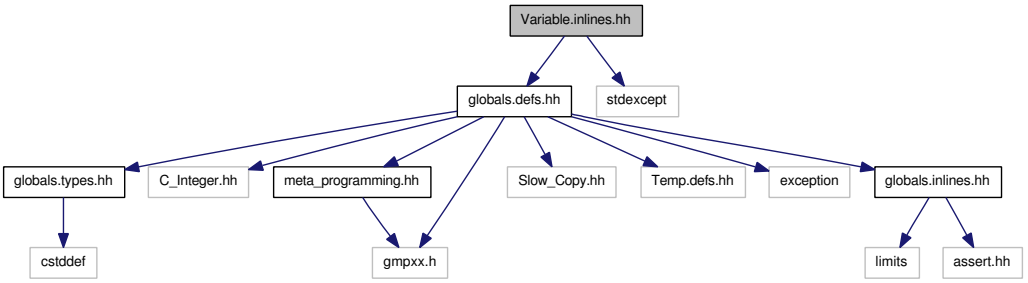
- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

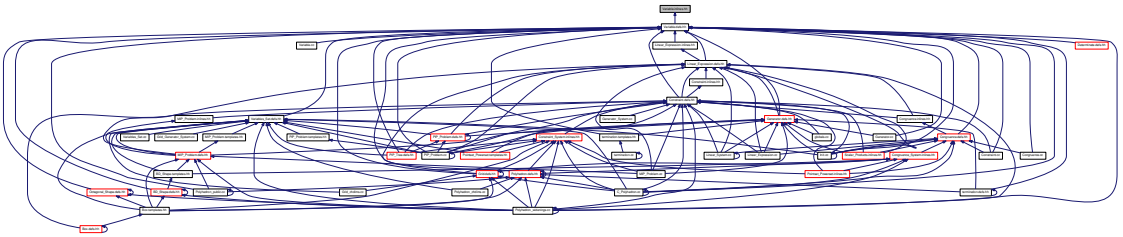
12.235 Variable.inlines.hh File Reference

```
#include "globals.defs.hh"
#include <stdexcept>
```

Include dependency graph for Variable.inlines.hh:



This graph shows which files directly or indirectly include this file:



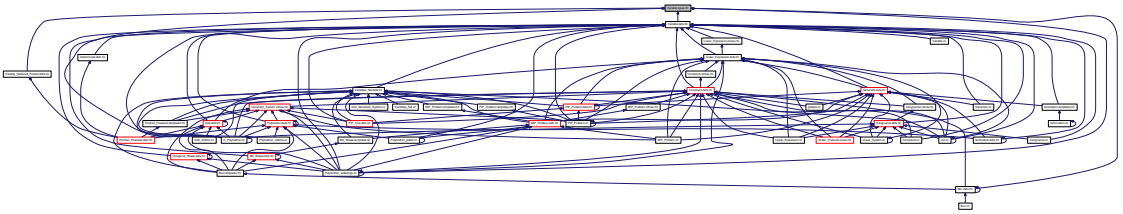
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.236 Variable.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

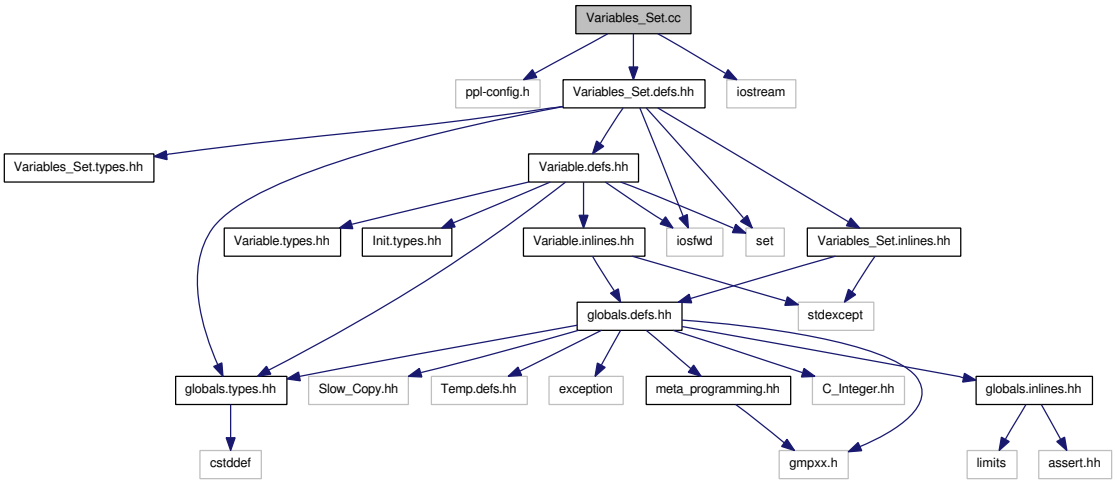
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.237 Variables_Set.cc File Reference

```
#include <ppl-config.h>
#include "Variables_Set.defs.hh"
#include <iostream>
```

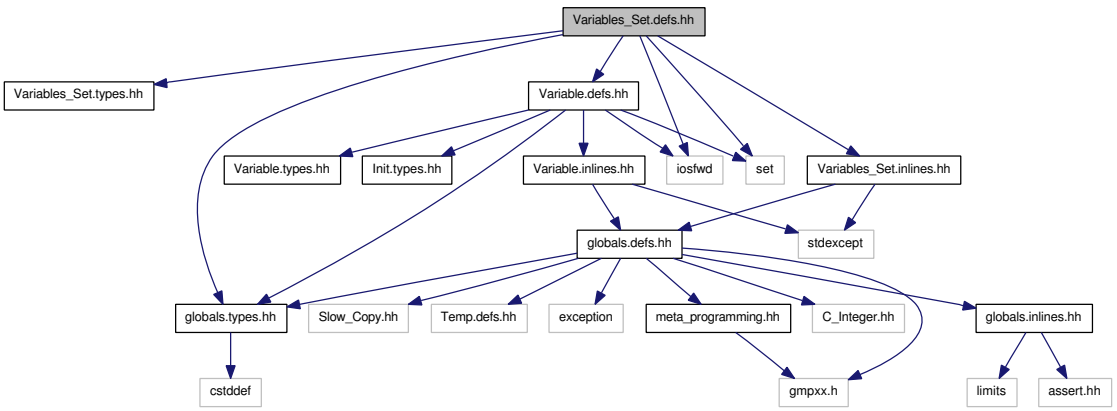
Include dependency graph for Variables_Set.cc:



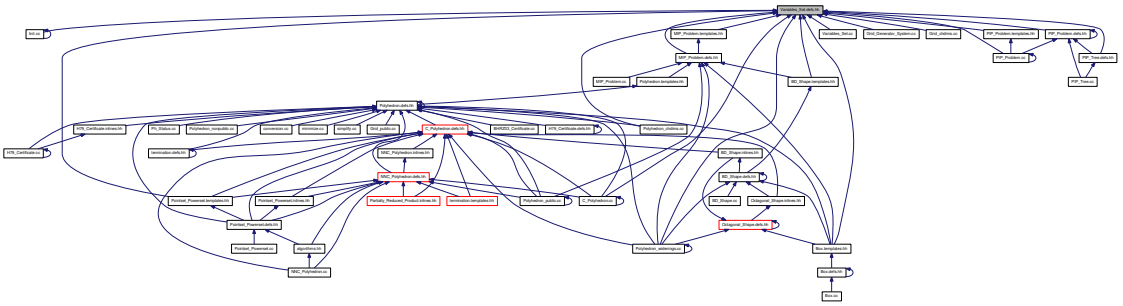
12.238 Variables_Set.defs.hh File Reference

```
#include "Variables_Set.types.hh"
#include "Variable.defs.hh"
#include "globals.types.hh"
#include <iosfwd>
#include <set>
#include "Variables_Set.inlines.hh"
```

Include dependency graph for Variables_Set.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Variables_Set

  An std::set of variables' indexes.

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

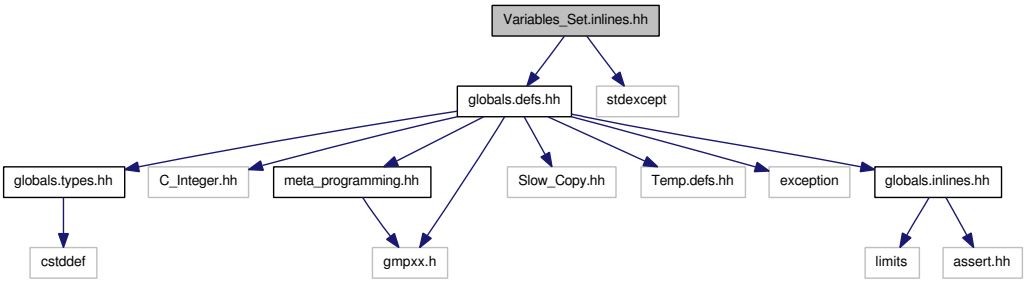
- namespace Parma_Polyhedra_Library::IO_Operators

  All input/output operators are confined to this namespace.

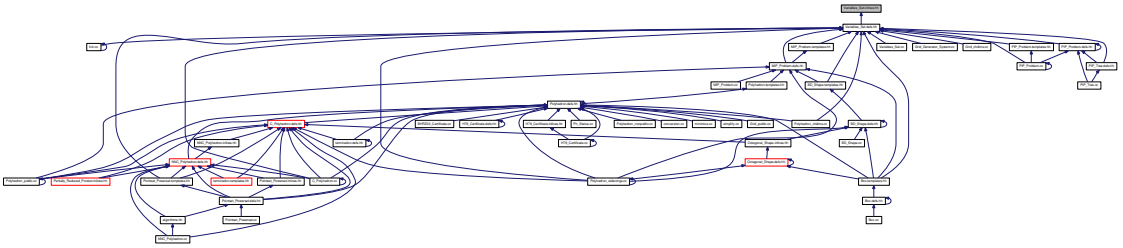
12.239 Variables_Set.inlines.hh File Reference

```
#include "globals.defs.hh"
#include <stdexcept>
```

Include dependency graph for Variables_Set.inlines.hh:



This graph shows which files directly or indirectly include this file:



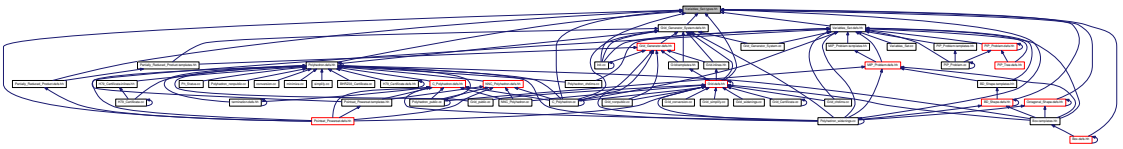
Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.240 Variables_Set.types.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

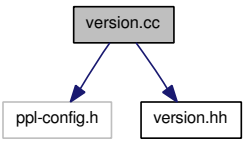
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.241 version.cc File Reference

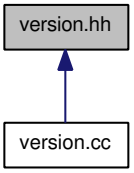
```
#include <ppl-config.h>
#include "version.hh"
```

Include dependency graph for version.cc:



12.242 version.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Defines

- #define PPL_VERSION_MAJOR 0

  The major number of the PPL version.

- #define PPL_VERSION_MINOR 11

  The minor number of the PPL version.

- #define PPL_VERSION_REVISION 2

  The revision number of the PPL version.

- #define PPL_VERSION_BETA 0

  The beta number of the PPL version. This is zero for official releases and nonzero for development snapshots.

- #define PPL_VERSION "0.11.2"

  A string containing the PPL version.

Functions

- unsigned Parma_Polyhedra_Library::version_major ()

  Returns the major number of the PPL version.

- unsigned Parma_Polyhedra_Library::version_minor ()

  Returns the minor number of the PPL version.

- unsigned Parma_Polyhedra_Library::version_revision ()

  Returns the revision number of the PPL version.

- unsigned Parma_Polyhedra_Library::version_beta ()

  Returns the beta number of the PPL version.

- const char * Parma_Polyhedra_Library::version ()

  Returns a character string containing the PPL version.

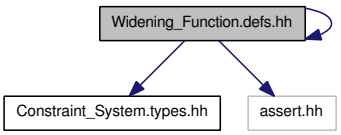
- const char * Parma_Polyhedra_Library::banner ()

  Returns a character string containing the PPL banner.

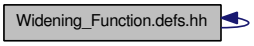
12.243 Widening_Function.defs.hh File Reference

```
#include "Widening_Function.types.hh"
#include "Constraint_System.types.hh"
#include "assert.hh"
```

Include dependency graph for Widening_Function.defs.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class Parma_Polyhedra_Library::Widening_Function< PSET >

  Wraps a widening method into a function object.

- class Parma_Polyhedra_Library::Limited_Widening_Function< PSET, CSYS >

  Wraps a limited widening method into a function object.

Namespaces

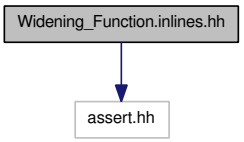
- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.244 Widening_Function.inlines.hh File Reference

```
#include "assert.hh"
```

Include dependency graph for Widening_Function.inlines.hh:



Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

12.245 Widening_Function.types.hh File Reference

Namespaces

- namespace Parma_Polyhedra_Library

  The entire library is confined to this namespace.

Index

~BD_Shape
Parma_Polyhedra_Library::BD_Shape, 276
~BHRZ03_Certificate
Parma_Polyhedra_Library::BHRZ03_-Certificate, 340
~Bit_Matrix
Parma_Polyhedra_Library::Bit_Matrix, 347
~Bit_Row
Parma_Polyhedra_Library::Bit_Row, 357
~C_Polyhedron
Parma_Polyhedra_Library::C_Polyhedron, 442
~Congruence
Parma_Polyhedra_Library::Congruence, 491
~Congruence_System
Parma_Polyhedra_Library::Congruence_-System, 510
~Congruences_Reduction
Parma_Polyhedra_Library::Congruences_-Reduction, 524
~Constraint
Parma_Polyhedra_Library::Constraint, 571
~Constraint_System
Parma_Polyhedra_Library::Constraint_-System, 593
~Constraints_Reduction
Parma_Polyhedra_Library::Constraints_-Reduction, 607
~DB_Matrix
Parma_Polyhedra_Library::DB_Matrix, 612
~DB_Row
Parma_Polyhedra_Library::DB_Row, 625
~DB_Row_Impl_Handler
Parma_Polyhedra_Library::DB_Row_Impl_-Handler, 634
~Determinate
Parma_Polyhedra_Library::Determinate, 639
~Generator
Parma_Polyhedra_Library::Generator, 709
~Generator_System
Parma_Polyhedra_Library::Generator_-System, 732
~Grid
Parma_Polyhedra_Library::Grid, 771
~Grid_Certificate
Parma_Polyhedra_Library::Grid_Certificate, 834
~Grid_Generator
Parma_Polyhedra_Library::Grid_Generator, 844
~Grid_Generator_System
Parma_Polyhedra_Library::Grid_Generator_-System, 864
~H79_Certificate
Parma_Polyhedra_Library::H79_Certificate, 878
~Impl
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 892
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 885
~Init
Parma_Polyhedra_Library::Init, 899
~Linear_Expression
Parma_Polyhedra_Library::Linear_-Expression, 1025
~Linear_Row
Parma_Polyhedra_Library::Linear_Row, 1046
~MIP_Problem
Parma_Polyhedra_Library::MIP_Problem, 1109
~Matrix
Parma_Polyhedra_Library::Matrix, 1082
~NNC_Polyhedron
Parma_Polyhedra_Library::NNC_Polyhedron, 1141
~No_Reduction
Parma_Polyhedra_Library::No_Reduction, 1144
~OR_Matrix
Parma_Polyhedra_Library::OR_Matrix, 1234
~Octagonal_Shape
Parma_Polyhedra_Library::Octagonal_Shape, 1164
~PIP_Decision_Node
Parma_Polyhedra_Library::PIP_Decision_-Node, 1291
~PIP_Problem
Parma_Polyhedra_Library::PIP_Problem, 1310
~PIP_Solution_Node
Parma_Polyhedra_Library::PIP_Solution_-Node, 1325
~PIP_Tree_Node
Parma_Polyhedra_Library::PIP_Tree_Node, 1338
~Partially_Reduced_Product
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1259
~Polyhedron

Parma_Polyhedra_Library::Polyhedron, 1435
~Powerset
Parma_Polyhedra_Library::Powerset, 1522
~Pseudo_Row
Parma_Polyhedra_Library::OR_-Matrix::Pseudo_Row, 1537
~RAII_Temporary_Real_Relaxation
Parma_Polyhedra_Library::MIP_-Problem::RAII_Temporary_Real_-Relaxation, 1540
~Rep
Parma_Polyhedra_Library::Determinate::Rep, 1544
~Row
Parma_Polyhedra_Library::Row, 1560
~Row_Impl_Handler
Parma_Polyhedra_Library::Row_Impl_-Handler, 1570
~Shape_Preserving_Reduction
Parma_Polyhedra_Library::Shape_-Preserving_Reduction, 1592
~Smash_Reduction
Parma_Polyhedra_Library::Smash_Reduction, 1599
~Tableau
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1655
~Throwable
Parma_Polyhedra_Library::Throwable, 1660
~const_iterator
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 541
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 536
Parma_Polyhedra_Library::Generator_-System::const_iterator, 546
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 552

abandon_expensive_computations
PPL_CXX_interface, 104
abs_assign
Parma_Polyhedra_Library, 122
Parma_Polyhedra_Library::Checked_Number, 465
Parma_Polyhedra_Library::GMP_Integer, 746
abs_float
Parma_Polyhedra_Library::Checked, 178
abs_generic
Parma_Polyhedra_Library::Checked, 178
abs_mpq
Parma_Polyhedra_Library::Checked, 178
abs_mpz
Parma_Polyhedra_Library::Checked, 178
add_2exp_float
Parma_Polyhedra_Library::Checked, 179
add_2exp_mpq
Parma_Polyhedra_Library::Checked, 179
add_2exp_mpz
Parma_Polyhedra_Library::Checked, 179
add_2exp_signed_int
Parma_Polyhedra_Library::Checked, 179
add_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 179
add_and_minimize
Parma_Polyhedra_Library::Polyhedron, 1435, 1436
add_assign
Parma_Polyhedra_Library::Interval, 907
add_congruence
Parma_Polyhedra_Library::BD_Shape, 277
Parma_Polyhedra_Library::Box, 383
Parma_Polyhedra_Library::Grid, 772
Parma_Polyhedra_Library::Octagonal_Shape, 1165
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1260
Parma_Polyhedra_Library::Pointset_-Powerset, 1366
Parma_Polyhedra_Library::Polyhedron, 1437
add_congruence_no_check
Parma_Polyhedra_Library::Box, 384
Parma_Polyhedra_Library::Grid, 772
add_congruences
Parma_Polyhedra_Library::BD_Shape, 277
Parma_Polyhedra_Library::Box, 384
Parma_Polyhedra_Library::Grid, 772
Parma_Polyhedra_Library::Octagonal_Shape, 1165
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1260
Parma_Polyhedra_Library::Pointset_-Powerset, 1366
Parma_Polyhedra_Library::Polyhedron, 1437
add_congruences_no_check
Parma_Polyhedra_Library::Box, 384
add_constraint
Parma_Polyhedra_Library::BD_Shape, 277
Parma_Polyhedra_Library::Box, 385
Parma_Polyhedra_Library::Grid, 773
Parma_Polyhedra_Library::Interval, 907
Parma_Polyhedra_Library::MIP_Problem, 1109
Parma_Polyhedra_Library::Octagonal_Shape, 1165
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1260

Parma_Polyhedra_Library::PIP_Problem, 1310
Parma_Polyhedra_Library::PIP_Tree_Node, 1338
Parma_Polyhedra_Library::Pointset_-Powerset, 1366
Parma_Polyhedra_Library::Polyhedron, 1438
add_constraint_no_check
Parma_Polyhedra_Library::Box, 385
Parma_Polyhedra_Library::Grid, 773
add_constraints
Parma_Polyhedra_Library::BD_Shape, 278
Parma_Polyhedra_Library::Box, 385
Parma_Polyhedra_Library::Grid, 773
Parma_Polyhedra_Library::MIP_Problem, 1109
Parma_Polyhedra_Library::Octagonal_Shape, 1166
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1261
Parma_Polyhedra_Library::PIP_Problem, 1310
Parma_Polyhedra_Library::Pointset_-Powerset, 1366
Parma_Polyhedra_Library::Polyhedron, 1438
add_constraints_no_check
Parma_Polyhedra_Library::Box, 386
add_corresponding_closure_points
Parma_Polyhedra_Library::Generator_-System, 733
add_corresponding_points
Parma_Polyhedra_Library::Generator_-System, 733
add_dbm_constraint
Parma_Polyhedra_Library::BD_Shape, 278, 279
add_disjunct
Parma_Polyhedra_Library::Pointset_-Powerset, 1367
Parma_Polyhedra_Library::Powerset, 1522
add_float
Parma_Polyhedra_Library::Checked, 179
add_generator
Parma_Polyhedra_Library::Polyhedron, 1439
add_generators
Parma_Polyhedra_Library::Polyhedron, 1439
add_grid_generator
Parma_Polyhedra_Library::Grid, 774
add_grid_generators
Parma_Polyhedra_Library::Grid, 774
add_int_larger
Parma_Polyhedra_Library::Checked, 180
add_interval_constraint_no_check
Parma_Polyhedra_Library::Box, 386
add_low_level_constraints
Parma_Polyhedra_Library::Constraint_-System, 594
add_mpq
Parma_Polyhedra_Library::Checked, 180
add_mpz
Parma_Polyhedra_Library::Checked, 180
add_mul_assign
Parma_Polyhedra_Library, 123
Parma_Polyhedra_Library::Checked_Number, 465
Parma_Polyhedra_Library::GMP_Integer, 746
Parma_Polyhedra_Library::Linear_-Expression, 1032
add_mul_float
Parma_Polyhedra_Library::Checked, 180
add_mul_int
Parma_Polyhedra_Library::Checked, 180
add_mul_mpq
Parma_Polyhedra_Library::Checked, 180
add_mul_mpz
Parma_Polyhedra_Library::Checked, 181
add_non_bottom_disjunct_preserve_reduction
Parma_Polyhedra_Library::Powerset, 1522, 1523
add_octagonal_constraint
Parma_Polyhedra_Library::Octagonal_Shape, 1166, 1167
add_pending_row
Parma_Polyhedra_Library::Linear_System, 1061, 1062
add_pending_rows
Parma_Polyhedra_Library::Linear_System, 1062
add_recycled_congruences
Parma_Polyhedra_Library::BD_Shape, 279
Parma_Polyhedra_Library::Box, 386
Parma_Polyhedra_Library::Grid, 775
Parma_Polyhedra_Library::Octagonal_Shape, 1167
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1261
Parma_Polyhedra_Library::Polyhedron, 1440
add_recycled_constraints
Parma_Polyhedra_Library::BD_Shape, 279
Parma_Polyhedra_Library::Box, 387
Parma_Polyhedra_Library::Grid, 775
Parma_Polyhedra_Library::Octagonal_Shape, 1167
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1261
Parma_Polyhedra_Library::Polyhedron, 1440
add_recycled_generators
Parma_Polyhedra_Library::Polyhedron, 1441

add_recycled_grid_generators
Parma_Polyhedra_Library::Grid, 776
add_recycled_row
Parma_Polyhedra_Library::Bit_Matrix, 347
Parma_Polyhedra_Library::Matrix, 1083
add_restriction
Parma_Polyhedra_Library, 123
add_row
Parma_Polyhedra_Library::Linear_System, 1062
Parma_Polyhedra_Library::Matrix, 1083
add_rows
Parma_Polyhedra_Library::Linear_System, 1063
add_rows_and_columns
Parma_Polyhedra_Library::Linear_System, 1063
add_signed_int
Parma_Polyhedra_Library::Checked, 181
add_space_dimensions
Parma_Polyhedra_Library::Grid, 776, 777
Parma_Polyhedra_Library::Polyhedron, 1442
add_space_dimensions_and_embed
Parma_Polyhedra_Library::BD_Shape, 280
Parma_Polyhedra_Library::Box, 387
Parma_Polyhedra_Library::Grid, 777
Parma_Polyhedra_Library::MIP_Problem, 1110
Parma_Polyhedra_Library::Octagonal_Shape, 1168
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1262
Parma_Polyhedra_Library::PIP_Problem, 1311
Parma_Polyhedra_Library::Pointset_-Powerset, 1367
Parma_Polyhedra_Library::Polyhedron, 1442
add_space_dimensions_and_project
Parma_Polyhedra_Library::BD_Shape, 280
Parma_Polyhedra_Library::Box, 388
Parma_Polyhedra_Library::Grid, 778
Parma_Polyhedra_Library::Octagonal_Shape, 1168
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1262
Parma_Polyhedra_Library::Pointset_-Powerset, 1367
Parma_Polyhedra_Library::Polyhedron, 1443
add_to_integer_space_dimensions
Parma_Polyhedra_Library::MIP_Problem, 1110
add_to_parameter_space_dimensions
Parma_Polyhedra_Library::PIP_Problem, 1311
add_unit_rows_and_columns
Parma_Polyhedra_Library::Congruence_-System, 510
add_universe_rows_and_columns
Parma_Polyhedra_Library::Grid_Generator_-System, 864
add_unsigned_int
Parma_Polyhedra_Library::Checked, 181
add_zero_columns
Parma_Polyhedra_Library::Matrix, 1083
add_zero_rows
Parma_Polyhedra_Library::Matrix, 1084
add_zero_rows_and_columns
Parma_Polyhedra_Library::Matrix, 1084
addmod
Parma_Polyhedra_Library, 124
adjust_topology_and_space_dimension
Parma_Polyhedra_Library::Constraint_-System, 594
Parma_Polyhedra_Library::Generator_-System, 733
affine_dim
Parma_Polyhedra_Library::BHRZ03_-Certificate, 341
Parma_Polyhedra_Library::H79_Certificate, 879
affine_dimension
Parma_Polyhedra_Library::BD_Shape, 281
Parma_Polyhedra_Library::Box, 388
Parma_Polyhedra_Library::Grid, 778
Parma_Polyhedra_Library::Octagonal_Shape, 1169
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1262
Parma_Polyhedra_Library::Pointset_-Powerset, 1368
Parma_Polyhedra_Library::Polyhedron, 1444
affine_image
Parma_Polyhedra_Library::BD_Shape, 281
Parma_Polyhedra_Library::Box, 388
Parma_Polyhedra_Library::Generator_-System, 734
Parma_Polyhedra_Library::Grid, 778
Parma_Polyhedra_Library::Grid_Generator_-System, 864
Parma_Polyhedra_Library::Octagonal_Shape, 1169
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1263
Parma_Polyhedra_Library::Pointset_-Powerset, 1368
Parma_Polyhedra_Library::Polyhedron, 1444
affine_preimage
Parma_Polyhedra_Library::BD_Shape, 282

Parma_Polyhedra_Library::Box, 389
Parma_Polyhedra_Library::Congruence_-System, 511
Parma_Polyhedra_Library::Constraint_-System, 595
Parma_Polyhedra_Library::Grid, 779
Parma_Polyhedra_Library::Octagonal_Shape, 1170
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1263
Parma_Polyhedra_Library::Pointset_-Powerset, 1368
Parma_Polyhedra_Library::Polyhedron, 1445
algorithms.hh, 1680
all_affine_quasi_ranking_functions_MS
Parma_Polyhedra_Library, 124
Parma_Polyhedra_-Library::Implementation::Termination, 227
all_affine_quasi_ranking_functions_MS_2
Parma_Polyhedra_Library, 125
all_affine_ranking_functions_MS
Parma_Polyhedra_Library, 125
Parma_Polyhedra_-Library::Implementation::Termination, 227
all_affine_ranking_functions_MS_2
Parma_Polyhedra_Library, 126
all_affine_ranking_functions_PR
Parma_Polyhedra_Library, 127
Parma_Polyhedra_-Library::Implementation::Termination, 227
all_affine_ranking_functions_PR_2
Parma_Polyhedra_Library, 127
all_affine_ranking_functions_PR_original
Parma_Polyhedra_-Library::Implementation::Termination, 227
all_homogeneous_terms_are_zero
Parma_Polyhedra_Library::Grid_Generator, 844
Parma_Polyhedra_Library::Linear_-Expression, 1027
Parma_Polyhedra_Library::Linear_Row, 1046
allocate
Parma_Polyhedra_Library::DB_Row, 626
Parma_Polyhedra_Library::Row, 1560
anonymous_enum
Parma_Polyhedra_Library::Bool, 365
Parma_Polyhedra_Library::Is_Same_Or_-Derived, 982
Any
Parma_Polyhedra_Library::Is_Same_Or_-Derived::Any, 242
ANY_COMPLEXITY
PPL_CXX_interface, 102
any_row_iterator
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 245, 250
Parma_Polyhedra_Library::OR_-Matrix::Pseudo_Row, 1538
approximate_partition
Parma_Polyhedra_Library::Pointset_-Powerset, 1395
approximate_partition_aux
Parma_Polyhedra_Library::Pointset_-Powerset, 1396
art_parameter_begin
Parma_Polyhedra_Library::PIP_Tree_Node, 1338
art_parameter_count
Parma_Polyhedra_Library::PIP_Tree_Node, 1338
art_parameter_end
Parma_Polyhedra_Library::PIP_Tree_Node, 1339
Artificial_Parameter
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 255
Artificial_Parameter_Sequence
Parma_Polyhedra_Library::PIP_Tree_Node, 1337
artificial_parameters
Parma_Polyhedra_Library::PIP_Tree_Node, 1346
as_decision
Parma_Polyhedra_Library::PIP_Decision_-Node, 1292
Parma_Polyhedra_Library::PIP_Tree_Node, 1339
as_solution
Parma_Polyhedra_Library::PIP_Solution_-Node, 1325
Parma_Polyhedra_Library::PIP_Tree_Node, 1339
ascii_dump
Parma_Polyhedra_Library, 127
Parma_Polyhedra_Library::BD_Shape, 282, 283
Parma_Polyhedra_Library::BD_-Shape::Status, 1618
Parma_Polyhedra_Library::Bit_Matrix, 347, 348
Parma_Polyhedra_Library::Box, 390
Parma_Polyhedra_Library::Box::Status, 1612
Parma_Polyhedra_Library::Congruence, 492

Parma_Polyhedra_Library::Congruence_-
System, 511, 512
Parma_Polyhedra_Library::Constraint, 572
Parma_Polyhedra_Library::Constraint_-
System, 595, 596
Parma_Polyhedra_Library::DB_Matrix, 612
Parma_Polyhedra_Library::Generator, 710
Parma_Polyhedra_Library::Generator_-
System, 735
Parma_Polyhedra_Library::Grid, 780, 781
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_Library::Grid_Generator,
844, 845
Parma_Polyhedra_Library::Grid_Generator_-
System, 865
Parma_Polyhedra_Library::Interval, 907
Parma_Polyhedra_Library::Interval_Info_-
Bitset, 932
Parma_Polyhedra_Library::Interval_Info_-
Null, 938
Parma_Polyhedra_Library::Interval_Info_-
Null_Open, 942
Parma_Polyhedra_Library::Linear_-
Expression, 1027
Parma_Polyhedra_Library::Linear_Row, 1046
Parma_Polyhedra_Library::Linear_-
Row::Flags, 664
Parma_Polyhedra_Library::Linear_System,
1063, 1064
Parma_Polyhedra_Library::Matrix, 1085
Parma_Polyhedra_Library::MIP_Problem,
1111
Parma_Polyhedra_Library::Octagonal_Shape,
1171
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1646
Parma_Polyhedra_Library::OR_Matrix, 1234
Parma_Polyhedra_Library::Partially_-
Reduced_Product, 1264
Parma_Polyhedra_Library::PIP_Decision_-
Node, 1292
Parma_Polyhedra_Library::PIP_Problem,
1312
Parma_Polyhedra_Library::PIP_Solution_-
Node, 1325
Parma_Polyhedra_Library::PIP_Solution_-
Node::Tableau, 1655
Parma_Polyhedra_Library::PIP_Tree_Node,
1339
Parma_Polyhedra_Library::PIP_Tree_-
Node::Artificial_Parameter, 256
Parma_Polyhedra_Library::Pointset_-
Powerset, 1369
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1402
Parma_Polyhedra_Library::Poly_Gen_-
Relation, 1410
Parma_Polyhedra_Library::Polyhedron, 1447
Parma_Polyhedra_-
Library::Polyhedron::Status, 1634
Parma_Polyhedra_Library::Row, 1560
Parma_Polyhedra_Library::Row::Flags, 658
Parma_Polyhedra_Library::Variables_Set,
1673
Status, 1625
ascii_load
Parma_Polyhedra_Library, 128
Parma_Polyhedra_Library::BD_Shape, 283
Parma_Polyhedra_Library::BD_-
Shape::Status, 1618
Parma_Polyhedra_Library::Bit_Matrix, 348
Parma_Polyhedra_Library::Box, 390
Parma_Polyhedra_Library::Box::Status, 1612
Parma_Polyhedra_Library::Congruence, 492
Parma_Polyhedra_Library::Congruence_-
System, 512
Parma_Polyhedra_Library::Constraint, 572
Parma_Polyhedra_Library::Constraint_-
System, 596
Parma_Polyhedra_Library::DB_Matrix, 612
Parma_Polyhedra_Library::Generator, 710
Parma_Polyhedra_Library::Generator_-
System, 735
Parma_Polyhedra_Library::Grid, 781
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_Library::Grid_Generator,
845
Parma_Polyhedra_Library::Grid_Generator_-
System, 865
Parma_Polyhedra_Library::Interval, 907
Parma_Polyhedra_Library::Interval_Info_-
Bitset, 932
Parma_Polyhedra_Library::Interval_Info_-
Null, 938
Parma_Polyhedra_Library::Interval_Info_-
Null_Open, 942
Parma_Polyhedra_Library::Linear_-
Expression, 1028
Parma_Polyhedra_Library::Linear_Row, 1047
Parma_Polyhedra_Library::Linear_-
Row::Flags, 664
Parma_Polyhedra_Library::Linear_System,
1064
Parma_Polyhedra_Library::Matrix, 1085
Parma_Polyhedra_Library::MIP_Problem,
1111

Parma_Polyhedra_Library::Octagonal_Shape, 1171
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1646
Parma_Polyhedra_Library::OR_Matrix, 1235
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1264
Parma_Polyhedra_Library::PIP_Decision_-Node, 1293
Parma_Polyhedra_Library::PIP_Problem, 1312
Parma_Polyhedra_Library::PIP_Solution_-Node, 1326
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1655
Parma_Polyhedra_Library::PIP_Tree_Node, 1340
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 256
Parma_Polyhedra_Library::Pointset_-Powerset, 1369
Parma_Polyhedra_Library::Polyhedron, 1447
Parma_Polyhedra_-Library::Polyhedron::Status, 1634
Parma_Polyhedra_Library::Row, 1561
Parma_Polyhedra_Library::Row::Flags, 658
Parma_Polyhedra_Library::Variables_Set, 1674
Status, 1625
assign
Parma_Polyhedra_Library::DB_Row, 626
Parma_Polyhedra_Library::Interval, 908
Parma_Polyhedra_Library::Row, 1561
Parma_Polyhedra_Library::Scalar_Products, 1579, 1580
assign_all_inequalities_approximation
Parma_Polyhedra_-Library::Implementation::Termination, 227, 228
assign_exact
Parma_Polyhedra_Library::Checked, 181
assign_float_float
Parma_Polyhedra_Library::Checked, 181
assign_float_float_exact
Parma_Polyhedra_Library::Checked, 181
assign_float_float_inexact
Parma_Polyhedra_Library::Checked, 182
assign_float_int
Parma_Polyhedra_Library::Checked, 182
assign_float_int_inexact
Parma_Polyhedra_Library::Checked, 182
assign_float_mpq
Parma_Polyhedra_Library::Checked, 182
assign_float_mpz
Parma_Polyhedra_Library::Checked, 182
assign_int_float
Parma_Polyhedra_Library::Checked, 183
assign_int_mpq
Parma_Polyhedra_Library::Checked, 183
assign_mpq_float
Parma_Polyhedra_Library::Checked, 183
assign_mpq_long_double
Parma_Polyhedra_Library::Checked, 183
assign_mpq_signed_int
Parma_Polyhedra_Library::Checked, 183
assign_mpq_unsigned_int
Parma_Polyhedra_Library::Checked, 183
assign_mpz_float
Parma_Polyhedra_Library::Checked, 184
assign_mpz_long_double
Parma_Polyhedra_Library::Checked, 184
assign_mpz_mpq
Parma_Polyhedra_Library::Checked, 184
assign_mpz_signed_int
Parma_Polyhedra_Library::Checked, 184
assign_mpz_unsigned_int
Parma_Polyhedra_Library::Checked, 184
assign_nan
Parma_Polyhedra_Library::Checked, 185
assign_or_swap
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 947
assign_r
Parma_Polyhedra_Library, 128
Parma_Polyhedra_Library::Checked_Number, 466
assign_rem
Parma_Polyhedra_Library, 129
assign_restriction
Parma_Polyhedra_Library, 129, 130
assign_signed_int_mpz
Parma_Polyhedra_Library::Checked, 185
assign_signed_int_signed_int
Parma_Polyhedra_Library::Checked, 185
assign_signed_int_unsigned_int
Parma_Polyhedra_Library::Checked, 185
assign_special_float
Parma_Polyhedra_Library::Checked, 185
assign_special_int
Parma_Polyhedra_Library::Checked, 185
assign_special_mpq
Parma_Polyhedra_Library::Checked, 186
assign_special_mpz
Parma_Polyhedra_Library::Checked, 186
assign_unsigned_int_mpz
Parma_Polyhedra_Library::Checked, 186
assign_unsigned_int_signed_int
Parma_Polyhedra_Library::Checked, 186

assign_unsigned_int_unsigned_int
Parma_Polyhedra_Library::Checked, 186
assign_with_pending
Parma_Polyhedra_Library::Linear_System, 1064

back_substitute
Parma_Polyhedra_Library::Linear_System, 1064
banner
Parma_Polyhedra_Library, 130
BASE
Parma_Polyhedra_Library::float_ibm_double, 671
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 697
Base
Parma_Polyhedra_Library::const_iterator_-to_const, 556
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Pointset_-Powerset, 1357
Parma_Polyhedra_Library::Variables_Set, 1673
base
Parma_Polyhedra_-Library::Checked::number_struct, 1145
Parma_Polyhedra_Library::const_iterator_-to_const, 560
Parma_Polyhedra_Library::iterator_to_const, 996
Parma_Polyhedra_Library::MIP_Problem, 1128
base_for_exponent
Parma_Polyhedra_-Library::Checked::number_struct, 1145
base_type
Parma_Polyhedra_Library::Row::Flags, 657
basis
Parma_Polyhedra_Library::PIP_Solution_-Node, 1331
BD_Shape
Parma_Polyhedra_Library::BD_Shape, 272–276
BD_Shape.cc, 1681
BD_Shape.defs.hh, 1681
PPL_IN_BD_Shape_CLASS, 1683
BD_Shape.inlines.hh, 1683
BD_Shape.templates.hh, 1684
BD_Shape.types.hh, 1686
BE_LAZY
Grid_nonpublic.cc, 1809
Polyhedron_chdims.cc, 1894
Polyhedron_nonpublic.cc, 1895
begin
Parma_Polyhedra_Library::Congruence_-System, 512
Parma_Polyhedra_Library::Constraint_-System, 596
Parma_Polyhedra_Library::DB_Matrix, 612
Parma_Polyhedra_Library::DB_Row, 626
Parma_Polyhedra_Library::Generator_-System, 736
Parma_Polyhedra_Library::Grid_Generator_-System, 866
Parma_Polyhedra_Library::Matrix, 1086
Parma_Polyhedra_Library::Powerset, 1523
BFT00_poly_hull_assign_if_exact
Parma_Polyhedra_Library::Polyhedron, 1447
BFT00_upper_bound_assign_if_exact
Parma_Polyhedra_Library::BD_Shape, 283
BGP99_extrapolation_assign
Parma_Polyhedra_Library::Pointset_-Powerset, 1370
BGP99_heuristics_assign
Parma_Polyhedra_Library::Pointset_-Powerset, 1370
BHMZ05_widening_assign
Parma_Polyhedra_Library::BD_Shape, 284
Parma_Polyhedra_Library::Octagonal_Shape, 1171
BHRZ03_Certificate
Parma_Polyhedra_Library::BHRZ03_-Certificate, 339
BHRZ03_Certificate.cc, 1686
BHRZ03_Certificate.defs.hh, 1686
BHRZ03_Certificate.inlines.hh, 1687
BHRZ03_Certificate.types.hh, 1688
BHRZ03_combining_constraints
Parma_Polyhedra_Library::Polyhedron, 1448
BHRZ03_evolving_points
Parma_Polyhedra_Library::Polyhedron, 1448
BHRZ03_evolving_rays
Parma_Polyhedra_Library::Polyhedron, 1448
BHRZ03_widening_assign
Parma_Polyhedra_Library::Polyhedron, 1449

BHZ03_widening_assign
Parma_Polyhedra_Library::Pointset_-Powerset, 1371
BHZ09_C_poly_hull_assign_if_exact
Parma_Polyhedra_Library::Polyhedron, 1449
BHZ09_NNC_poly_hull_assign_if_exact
Parma_Polyhedra_Library::Polyhedron, 1449
BHZ09_poly_hull_assign_if_exact
Parma_Polyhedra_Library::Polyhedron, 1450
BHZ09_upper_bound_assign_if_exact
Parma_Polyhedra_Library::BD_Shape, 284
big_dimension
Parma_Polyhedra_Library::PIP_Solution_-Node, 1331
big_parameter_dimension
Parma_Polyhedra_Library::PIP_Problem, 1319
Binary_Operator_Assign_Lifter
Parma_Polyhedra_-Library::Determinate::Binary_Operator_-Assign_Lifter, 343
Bit_Matrix
Parma_Polyhedra_Library::Bit_Matrix, 347
Bit_Matrix.cc, 1688
Bit_Matrix.defs.hh, 1689
Bit_Matrix.inlines.hh, 1690
Bit_Matrix.types.hh, 1691
Bit_Row
Parma_Polyhedra_Library::Bit_Row, 357
Bit_Row.cc, 1691
Bit_Row.defs.hh, 1691
Bit_Row.inlines.hh, 1692
PPL_BITS_PER_GMP_LIMB, 1694
Bit_Row.types.hh, 1694
bits
Parma_Polyhedra_Library::Row::Flags, 660
BITS_128
PPL_CXX_interface, 102
BITS_16
PPL_CXX_interface, 102
BITS_32
PPL_CXX_interface, 102
BITS_64
PPL_CXX_interface, 102
BITS_8
PPL_CXX_interface, 102
bitset
Parma_Polyhedra_Library::Interval_Info_-Bitset, 937
bool
Parma_Polyhedra_Library::Checked, 220
boundary_type
Parma_Polyhedra_Library::Interval, 906
bounded_affine_image
Parma_Polyhedra_Library::BD_Shape, 285
Parma_Polyhedra_Library::Box, 390
Parma_Polyhedra_Library::Grid, 781
Parma_Polyhedra_Library::Octagonal_Shape, 1172
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1264
Parma_Polyhedra_Library::Pointset_-Powerset, 1371
Parma_Polyhedra_Library::Polyhedron, 1450
bounded_affine_preimage
Parma_Polyhedra_Library::BD_Shape, 285
Parma_Polyhedra_Library::Box, 391
Parma_Polyhedra_Library::Grid, 782
Parma_Polyhedra_Library::Octagonal_Shape, 1173
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1265
Parma_Polyhedra_Library::Pointset_-Powerset, 1372
Parma_Polyhedra_Library::Polyhedron, 1451
bounded_BHRZ03_extrapolation_assign
Parma_Polyhedra_Library::Polyhedron, 1451
bounded_H79_extrapolation_assign
Parma_Polyhedra_Library::Polyhedron, 1452
Bounded_Integer_Type_Overflow
PPL_CXX_interface, 101
Bounded_Integer_Type_Representation
PPL_CXX_interface, 101
Bounded_Integer_Type_Width
PPL_CXX_interface, 101
bounds
Parma_Polyhedra_Library::BD_Shape, 286
Parma_Polyhedra_Library::Box, 391
Parma_Polyhedra_Library::Grid, 782
Parma_Polyhedra_Library::Octagonal_Shape, 1173
Parma_Polyhedra_Library::Polyhedron, 1452
bounds_from_above
Parma_Polyhedra_Library::BD_Shape, 287
Parma_Polyhedra_Library::Box, 392
Parma_Polyhedra_Library::Grid, 783
Parma_Polyhedra_Library::Octagonal_Shape, 1174
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1265
Parma_Polyhedra_Library::Pointset_-Powerset, 1372
Parma_Polyhedra_Library::Polyhedron, 1453
bounds_from_below
Parma_Polyhedra_Library::BD_Shape, 287
Parma_Polyhedra_Library::Box, 392
Parma_Polyhedra_Library::Grid, 783

Parma_Polyhedra_Library::Octagonal_Shape, 1174
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1265
Parma_Polyhedra_Library::Pointset_-Powerset, 1373
Parma_Polyhedra_Library::Polyhedron, 1453
bounds_no_check
Parma_Polyhedra_Library::Grid, 783
Box
Parma_Polyhedra_Library::Box, 378–383
Box.cc, 1694
Box.defs.hh, 1694
PPL_IN_Box_CLASS, 1696
Box.inlines.hh, 1696
Box.templates.hh, 1697
Box.types.hh, 1699
build
Parma_Polyhedra_Library::float_ibm_single, 672
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 686
Parma_Polyhedra_Library::float_ieee754_-single, 691
Parma_Polyhedra_Library::float_intel_-double_extended, 696
Parma_Polyhedra_Library::Interval, 908, 909
bump_size
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 893
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 885

C++ Language Interface, 89
C_MINIMIZED
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
Status, 1629
C_Polyhedron
Parma_Polyhedra_Library::C_Polyhedron, 438–442
C_Polyhedron.cc, 1699
USE_BFT00, 1701
USE_BHZ09, 1701
C_Polyhedron.defs.hh, 1701
C_Polyhedron.inlines.hh, 1702
C_Polyhedron.types.hh, 1703
C_UP_TO_DATE
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
Status, 1629
can_be_exactly_joined_to
Parma_Polyhedra_Library::Interval, 909
can_have_something_pending
Parma_Polyhedra_Library::Polyhedron, 1453
can_recycle_congruence_systems
Parma_Polyhedra_Library::BD_Shape, 287
Parma_Polyhedra_Library::Box, 392
Parma_Polyhedra_Library::Grid, 784
Parma_Polyhedra_Library::Octagonal_Shape, 1175
Parma_Polyhedra_Library::Polyhedron, 1453
can_recycle_constraint_systems
Parma_Polyhedra_Library::BD_Shape, 287
Parma_Polyhedra_Library::Box, 392
Parma_Polyhedra_Library::Grid, 784
Parma_Polyhedra_Library::Octagonal_Shape, 1175
Parma_Polyhedra_Library::Polyhedron, 1454
CARDINALITY_0_
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
CARDINALITY_1_
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
CARDINALITY_IS_
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
CARDINALITY_0
Parma_Polyhedra_Library::Interval_NS, 239
CARDINALITY_1
Parma_Polyhedra_Library::Interval_NS, 239
CARDINALITY_IS
Parma_Polyhedra_Library::Interval_NS, 240
CC76_extrapolation_assign
Parma_Polyhedra_Library::BD_Shape, 287, 288
Parma_Polyhedra_Library::Octagonal_Shape, 1175, 1176
CC76_narrowing_assign
Parma_Polyhedra_Library::BD_Shape, 288
Parma_Polyhedra_Library::Box, 393
Parma_Polyhedra_Library::Octagonal_Shape, 1176
CC76_widening_assign
Parma_Polyhedra_Library::Box, 393, 394
Parma_Polyhedra_Library::Interval, 909
ceil_assign
Parma_Polyhedra_Library::Checked_Number, 466
ceil_float
Parma_Polyhedra_Library::Checked, 186

ceil_mpq
Parma_Polyhedra_Library::Checked, 187
char_type
Parma_Polyhedra_Library::stdiobuf, 1650
check_containment
Parma_Polyhedra_Library::Pointset_-Powerset, 1396, 1397
check_empty
Parma_Polyhedra_Library::Box, 394
Parma_Polyhedra_Library::Interval, 910
check_empty_arg
Parma_Polyhedra_Library, 130
check_function
Parma_Polyhedra_Library::Weightwatch_-Traits, 1678
check_omega_reduced
Parma_Polyhedra_Library::Powerset, 1524
check_ownership
Parma_Polyhedra_Library::PIP_Decision_-Node, 1293
Parma_Polyhedra_Library::PIP_Solution_-Node, 1326
Parma_Polyhedra_Library::PIP_Tree_Node, 1340
CHECK_P
checked.defs.hh, 1709
check_result
Parma_Polyhedra_Library, 130
check_sorted
Parma_Polyhedra_Library::Bit_Matrix, 348
Parma_Polyhedra_Library::Linear_System, 1065
check_strong_normalized
Parma_Polyhedra_Library::Linear_Row, 1047
checked.cc, 1703
checked.defs.hh, 1704
CHECK_P, 1709
nonconst, 1710
PPL_DECLARE_FUN1_0_0, 1710
PPL_DECLARE_FUN1_0_1, 1710
PPL_DECLARE_FUN1_0_2, 1710
PPL_DECLARE_FUN1_0_3, 1710
PPL_DECLARE_FUN1_1_1, 1710
PPL_DECLARE_FUN1_1_2, 1710
PPL_DECLARE_FUN1_2_2, 1710
PPL_DECLARE_FUN2_0_0, 1711
PPL_DECLARE_FUN2_0_1, 1711
PPL_DECLARE_FUN2_0_2, 1711
PPL_DECLARE_FUN3_0_1, 1711
PPL_DECLARE_FUN5_0_1, 1711
PPL_FUNCTION_CLASS, 1711
PPL_SPECIALIZE_ABS, 1711
PPL_SPECIALIZE_ADD, 1711
PPL_SPECIALIZE_ADD_2EXP, 1712
PPL_SPECIALIZE_ADD_MUL, 1712
PPL_SPECIALIZE_ASSIGN, 1712
PPL_SPECIALIZE_ASSIGN_SPECIAL, 1712
PPL_SPECIALIZE_CEIL, 1712
PPL_SPECIALIZE_CLASSIFY, 1712
PPL_SPECIALIZE_CMP, 1713
PPL_SPECIALIZE_CONSTRUCT, 1713
PPL_SPECIALIZE_CONSTRUCT_-SPECIAL, 1713
PPL_SPECIALIZE_COPY, 1713
PPL_SPECIALIZE_DIV, 1713
PPL_SPECIALIZE_DIV_2EXP, 1713
PPL_SPECIALIZE_FLOOR, 1714
PPL_SPECIALIZE_FUN1_0_0, 1714
PPL_SPECIALIZE_FUN1_0_1, 1714
PPL_SPECIALIZE_FUN1_0_2, 1714
PPL_SPECIALIZE_FUN1_0_3, 1714
PPL_SPECIALIZE_FUN1_1_1, 1714
PPL_SPECIALIZE_FUN1_1_2, 1714
PPL_SPECIALIZE_FUN1_2_2, 1715
PPL_SPECIALIZE_FUN2_0_0, 1715
PPL_SPECIALIZE_FUN2_0_1, 1715
PPL_SPECIALIZE_FUN2_0_2, 1715
PPL_SPECIALIZE_FUN3_0_1, 1715
PPL_SPECIALIZE_FUN5_0_1, 1715
PPL_SPECIALIZE_GCD, 1715
PPL_SPECIALIZE_GCDEXT, 1716
PPL_SPECIALIZE_IDIV, 1716
PPL_SPECIALIZE_INPUT, 1716
PPL_SPECIALIZE_IS_INT, 1716
PPL_SPECIALIZE_IS_MINF, 1716
PPL_SPECIALIZE_IS_NAN, 1716
PPL_SPECIALIZE_IS_PINF, 1716
PPL_SPECIALIZE_LCM, 1717
PPL_SPECIALIZE_MUL, 1717
PPL_SPECIALIZE_MUL_2EXP, 1717
PPL_SPECIALIZE_NEG, 1717
PPL_SPECIALIZE_OUTPUT, 1717
PPL_SPECIALIZE_REM, 1717
PPL_SPECIALIZE_SGN, 1718
PPL_SPECIALIZE_SMOD_2EXP, 1718
PPL_SPECIALIZE_SQRT, 1718
PPL_SPECIALIZE_SUB, 1718
PPL_SPECIALIZE_SUB_2EXP, 1718
PPL_SPECIALIZE_SUB_MUL, 1718
PPL_SPECIALIZE_TRUNC, 1719
PPL_SPECIALIZE_UMOD_2EXP, 1719
checked.inlines.hh, 1719
PPL_GT_SILENT, 1723
PPL_LT_SILENT, 1723
PPL_MATCH_GMP_GCDEXT, 1723
PPL_SAFE_CONVERSION, 1723
checked_float.inlines.hh, 1723

checked_int.inlines.hh, 1727
PPL_ASSIGN2_SIGNED_SIGNED, 1732
PPL_ASSIGN2_SIGNED_UNSIGNED, 1732
PPL_ASSIGN2_UNSIGNED_SIGNED, 1732
PPL_ASSIGN2_UNSIGNED_UNSIGNED, 1732
PPL_ASSIGN_SIGNED, 1732
PPL_ASSIGN_UNSIGNED, 1732
strtoll, 1733
strtoull, 1733
checked_mpq.inlines.hh, 1733
checked_mpz.inlines.hh, 1737
Checked_Number
Parma_Polyhedra_Library::Checked_Number, 454–460
Checked_Number.cc, 1740
Checked_Number.defs.hh, 1740
PPL_DECLARE_FUNC1_A, 1743
PPL_DECLARE_FUNC1_B, 1743
PPL_DECLARE_FUNC2, 1743
PPL_DECLARE_FUNC4, 1744
Checked_Number.inlines.hh, 1744
DEBUG_ROUND_NOT_NEEDED, 1746
PPL_COND, 1746
PPL_COND_, 1746
PPL_COND_0, 1746
PPL_COND_1, 1746
PPL_DEFINE_ASSIGN_2EXP, 1746
PPL_DEFINE_ASSIGN_FUN2_1, 1746
PPL_DEFINE_ASSIGN_FUN2_2, 1746
PPL_DEFINE_ASSIGN_FUN3_3, 1747
PPL_DEFINE_ASSIGN_FUN5_5, 1747
PPL_DEFINE_BINARY_OP, 1747
PPL_DEFINE_BINARY_OP_ASSIGN, 1747
PPL_DEFINE_COMPARE, 1747
PPL_DEFINE_COMPARE_OP, 1747
PPL_DEFINE_CTOR, 1747
PPL_DEFINE_FUNC1_A, 1747
PPL_DEFINE_FUNC1_B, 1748
PPL_DEFINE_FUNC2, 1748
PPL_DEFINE_FUNC4, 1748
PPL_DEFINE_INCREMENT, 1748
Checked_Number.types.hh, 1748
child_node
Parma_Polyhedra_Library::PIP_Decision_Node, 1293
choose_branching_variable
Parma_Polyhedra_Library::MIP_Problem, 1111
classify
Parma_Polyhedra_Library::Checked_Number, 461
classify_float
Parma_Polyhedra_Library::Checked, 187
classify_int
Parma_Polyhedra_Library::Checked, 187
classify_mpq
Parma_Polyhedra_Library::Checked, 187
classify_mpz
Parma_Polyhedra_Library::Checked, 187
clear
Parma_Polyhedra_Library::Bit_Matrix, 348
Parma_Polyhedra_Library::Bit_Row, 357
Parma_Polyhedra_Library::Congruence_System, 513
Parma_Polyhedra_Library::Constraint_System, 597
Parma_Polyhedra_Library::Generator_System, 736
Parma_Polyhedra_Library::Grid_Generator_System, 866
Parma_Polyhedra_Library::Interval_Info_Bitset, 932
Parma_Polyhedra_Library::Interval_Info_Null, 938
Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo, 947
Parma_Polyhedra_Library::Linear_System, 1065
Parma_Polyhedra_Library::Matrix, 1086
Parma_Polyhedra_Library::MIP_Problem, 1112
Parma_Polyhedra_Library::OR_Matrix, 1235
Parma_Polyhedra_Library::PIP_Problem, 1312
Parma_Polyhedra_Library::Powerset, 1524
clear_boundary_properties
Parma_Polyhedra_Library::Interval_Info_Bitset, 933
Parma_Polyhedra_Library::Interval_Info_Null, 938
clear_congruences_minimized
Parma_Polyhedra_Library::Grid, 784
clear_congruences_up_to_date
Parma_Polyhedra_Library::Grid, 784
clear_constraints_minimized
Parma_Polyhedra_Library::Polyhedron, 1454
clear_constraints_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1454
clear_empty
Parma_Polyhedra_Library::Grid, 784
Parma_Polyhedra_Library::Polyhedron, 1454
clear_from
Parma_Polyhedra_Library::Bit_Row, 358
clear_generators_minimized
Parma_Polyhedra_Library::Grid, 785
Parma_Polyhedra_Library::Polyhedron, 1455
clear_generators_up_to_date

Parma_Polyhedra_Library::Grid, 785
Parma_Polyhedra_Library::Polyhedron, 1455
clear_pending_constraints
Parma_Polyhedra_Library::Polyhedron, 1455
clear_pending_generators
Parma_Polyhedra_Library::Polyhedron, 1455
clear_reduced_flag
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1266
clear_sat_c_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1456
clear_sat_g_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1456
clone
Parma_Polyhedra_Library::PIP_Decision_-Node, 1294
Parma_Polyhedra_Library::PIP_Solution_-Node, 1326
Parma_Polyhedra_Library::PIP_Tree_Node, 1340
CLOSURE_POINT
Parma_Polyhedra_Library::Generator, 709
closure_point
Parma_Polyhedra_Library::Generator, 711
cmp
Parma_Polyhedra_Library, 131
Parma_Polyhedra_Library::Checked_Number, 466
Parma_Polyhedra_Library::GMP_Integer, 747
cmp_float
Parma_Polyhedra_Library::Checked, 187
cmp_generic
Parma_Polyhedra_Library::Checked, 188
cmp_mp
Parma_Polyhedra_Library::Checked, 188
Coefficient
PPL_CXX_interface, 100
coefficient
Parma_Polyhedra_Library::Congruence, 493
Parma_Polyhedra_Library::Constraint, 573
Parma_Polyhedra_Library::Generator, 711
Parma_Polyhedra_Library::Grid_Generator, 845
Parma_Polyhedra_Library::Linear_-Expression, 1028
Parma_Polyhedra_Library::Linear_Row, 1047
Coefficient.cc, 1749
Coefficient.defs.hh, 1750
Coefficient.inlines.hh, 1751
Coefficient.types.hh, 1751
Coefficient_constants_finalize
Parma_Polyhedra_Library, 131
Coefficient_constants_initialize
Parma_Polyhedra_Library, 131
Coefficient_one
Parma_Polyhedra_Library, 131
coefficient_swap
Parma_Polyhedra_Library::Grid_Generator, 845
Coefficient_traits
PPL_CXX_interface, 100
Coefficient_traits_template.hh, 1752
coefficient_type
Parma_Polyhedra_Library::BD_Shape, 272
Parma_Polyhedra_Library::Grid, 765
Parma_Polyhedra_Library::Octagonal_Shape, 1159
Parma_Polyhedra_Library::Polyhedron, 1431
coefficient_type_base
Parma_Polyhedra_Library::BD_Shape, 272
Parma_Polyhedra_Library::Octagonal_Shape, 1159
Coefficient_zero
Parma_Polyhedra_Library, 131
coherent_index
Parma_Polyhedra_Library::Octagonal_Shape, 1219
collapse
Parma_Polyhedra_Library::Powerset, 1524, 1525
collect_certificates
Parma_Polyhedra_Library::Pointset_-Powerset, 1373
columns_erase_to_end
Parma_Polyhedra_Library::Bit_Matrix, 348
combine
Parma_Polyhedra_Library, 131
Parma_Polyhedra_Library::Euclidean_-Distance_Specialization, 648
Parma_Polyhedra_Library::L_Infinity_-Distance_Specialization, 997
Parma_Polyhedra_Library::Rectilinear_-Distance_Specialization, 1541
compare
Parma_Polyhedra_Library::BHRZ03_-Certificate, 340
Parma_Polyhedra_Library::Bit_Row, 361
Parma_Polyhedra_Library::Grid_Certificate, 834
Parma_Polyhedra_Library::H79_Certificate, 878
Parma_Polyhedra_Library::Linear_Row, 1053
compatibility_check
Parma_Polyhedra_Library::PIP_Tree_Node, 1340
Complexity_Class
PPL_CXX_interface, 102
compute_capacity

Parma_Polyhedra_Library, 132
compute_generator
Parma_Polyhedra_Library::MIP_Problem, 1112
compute_leader_indices
Parma_Polyhedra_Library::BD_Shape, 331
compute_leaders
Parma_Polyhedra_Library::BD_Shape, 289
Parma_Polyhedra_Library::Octagonal_Shape, 1176, 1177
compute_predecessors
Parma_Polyhedra_Library::BD_Shape, 289
compute_simplex_using_exact_pricing
Parma_Polyhedra_Library::MIP_Problem, 1112
compute_simplex_using_steepest_edge_float
Parma_Polyhedra_Library::MIP_Problem, 1113
compute_successors
Parma_Polyhedra_Library::Octagonal_Shape, 1177
compute_tableau
Parma_Polyhedra_Library::MIP_Problem, 1113
CON_VIRTUAL
Parma_Polyhedra_Library::Grid, 766
con_sys
Parma_Polyhedra_Library::Grid, 830
Parma_Polyhedra_Library::Polyhedron, 1511
concatenate
Parma_Polyhedra_Library::Congruence_System, 513
concatenate_assign
Parma_Polyhedra_Library::BD_Shape, 290
Parma_Polyhedra_Library::Box, 394
Parma_Polyhedra_Library::Determinate, 639
Parma_Polyhedra_Library::Grid, 785
Parma_Polyhedra_Library::Octagonal_Shape, 1177
Parma_Polyhedra_Library::Partially_Reduced_Product, 1266
Parma_Polyhedra_Library::Pointset_Powerset, 1373
Parma_Polyhedra_Library::Polyhedron, 1456
Congruence
Parma_Polyhedra_Library::Congruence, 491, 492
Congruence.cc, 1752
Congruence.defs.hh, 1753
Congruence.inlines.hh, 1754
Congruence.types.hh, 1755
Congruence_System
Parma_Polyhedra_Library::Congruence_System, 509, 510
Parma_Polyhedra_Library::Congruence_System::const_iterator, 543
Parma_Polyhedra_Library::Grid_Generator, 855
Congruence_System.cc, 1755
Congruence_System.defs.hh, 1756
Congruence_System.inlines.hh, 1757
Congruence_System.types.hh, 1758
congruence_widening_assign
Parma_Polyhedra_Library::Grid, 785
congruences
Parma_Polyhedra_Library::BD_Shape, 290
Parma_Polyhedra_Library::Box, 395
Parma_Polyhedra_Library::Grid, 786
Parma_Polyhedra_Library::Octagonal_Shape, 1178
Parma_Polyhedra_Library::Partially_Reduced_Product, 1266
Parma_Polyhedra_Library::Polyhedron, 1457
congruences_are_minimized
Parma_Polyhedra_Library::Grid, 786
congruences_are_up_to_date
Parma_Polyhedra_Library::Grid, 787
Congruences_Product
Parma_Polyhedra_Library::Domain_Product, 645
Congruences_Reduction
Parma_Polyhedra_Library::Congruences_Reduction, 524
const
Parma_Polyhedra_Library::Checked, 220
const_bool_nodef
Parma_Polyhedra_Library::Check_Overflow_Policy, 444–446
Parma_Polyhedra_Library::Checked::Float_2exp, 670
Parma_Polyhedra_Library::Checked::Larger< char >, 999
Parma_Polyhedra_Library::Checked::Larger< signed char >, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1002
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1003, 1004
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1005
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1007
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1008
Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1010
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1011, 1012

Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1013
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1015
Parma_Polyhedra_Library::Checked_Number_Transparent_Policy, 477, 478
Parma_Polyhedra_Library::Extended_Number_Policy, 651, 652
Parma_Polyhedra_Library::Interval_Info_Bitset, 933, 934
Parma_Polyhedra_Library::Interval_Info_Null, 939
Parma_Polyhedra_Library::Interval_Info_Null_Open, 942
Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy, 1577
PPL_CXX_interface, 98
const_element_iterator
Parma_Polyhedra_Library::OR_Matrix, 1232
const_int_nodef
Parma_Polyhedra_Library::Interval_Info_Bitset, 934, 935
Parma_Polyhedra_Library::Interval_Restriction_Integer, 944
PPL_CXX_interface, 98
const_iterator
Parma_Polyhedra_Library::Congruence_System, 521
Parma_Polyhedra_Library::Congruence_System::const_iterator, 541
Parma_Polyhedra_Library::Constraint_System, 605
Parma_Polyhedra_Library::Constraint_System::const_iterator, 536
Parma_Polyhedra_Library::DB_Matrix::const_iterator, 532
Parma_Polyhedra_Library::DB_Row, 624
Parma_Polyhedra_Library::Generator_System, 744
Parma_Polyhedra_Library::Generator_System::const_iterator, 546
Parma_Polyhedra_Library::Grid_Generator_System::const_iterator, 551, 552
Parma_Polyhedra_Library::Matrix::const_iterator, 527
Parma_Polyhedra_Library::MIP_Problem, 1105
Parma_Polyhedra_Library::PIP_Problem, 1308
Parma_Polyhedra_Library::Pointset_Powerset, 1357
Parma_Polyhedra_Library::Powerset, 1520
const_iterator_to_const
Parma_Polyhedra_Library::const_iterator_to_const, 557
const_iterator_to_const< Container >
Parma_Polyhedra_Library::iterator_to_const, 996
const_ref_nodef
PPL_CXX_interface, 98
const_reference
Parma_Polyhedra_Library::Coefficient_traits_template< GMP_Integer >, 481
const_reverse_iterator
Parma_Polyhedra_Library::Pointset_Powerset, 1357
Parma_Polyhedra_Library::Powerset, 1520
const_row_iterator
Parma_Polyhedra_Library::OR_Matrix, 1232
const_row_reference_type
Parma_Polyhedra_Library::OR_Matrix, 1233
const_value_nodef
PPL_CXX_interface, 98
constrains
Parma_Polyhedra_Library::BD_Shape, 290
Parma_Polyhedra_Library::Box, 395
Parma_Polyhedra_Library::Grid, 787
Parma_Polyhedra_Library::Octagonal_Shape, 1178
Parma_Polyhedra_Library::Partially_Reduced_Product, 1267
Parma_Polyhedra_Library::Pointset_Powerset, 1374
Parma_Polyhedra_Library::Polyhedron, 1457
Constraint
Parma_Polyhedra_Library::Constraint, 571, 572
Constraint.cc, 1758
Constraint.defs.hh, 1759
Constraint.inlines.hh, 1760
Constraint.types.hh, 1761
Constraint_Sequence
Parma_Polyhedra_Library::MIP_Problem, 1105
Parma_Polyhedra_Library::PIP_Problem, 1308
Parma_Polyhedra_Library::PIP_Tree_Node, 1337
Constraint_System
Parma_Polyhedra_Library::Constraint_System, 593, 594
Parma_Polyhedra_Library::Constraint_System::const_iterator, 538
Constraint_System.cc, 1761
Constraint_System.defs.hh, 1761
Constraint_System.inlines.hh, 1763
Constraint_System.types.hh, 1763

constraints
Parma_Polyhedra_Library::BD_Shape, 291
Parma_Polyhedra_Library::Box, 395
Parma_Polyhedra_Library::Grid, 787
Parma_Polyhedra_Library::Octagonal_Shape, 1178
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1267
Parma_Polyhedra_Library::PIP_Tree_Node, 1341
Parma_Polyhedra_Library::Polyhedron, 1457
constraints_
Parma_Polyhedra_Library::PIP_Tree_Node, 1346
constraints_are_minimized
Parma_Polyhedra_Library::Polyhedron, 1458
constraints_are_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1458
constraints_begin
Parma_Polyhedra_Library::MIP_Problem, 1113
Parma_Polyhedra_Library::PIP_Problem, 1313
constraints_end
Parma_Polyhedra_Library::MIP_Problem, 1113
Parma_Polyhedra_Library::PIP_Problem, 1313
Constraints_Product
Parma_Polyhedra_Library::Domain_Product, 645
Constraints_Reduction
Parma_Polyhedra_Library::Constraints_-Reduction, 607
construct
Parma_Polyhedra_Library, 132
Parma_Polyhedra_Library::Checked_Number, 467
Parma_Polyhedra_Library::DB_Row, 627
Parma_Polyhedra_Library::Grid, 787, 788
Parma_Polyhedra_Library::Linear_Row, 1047, 1048
Parma_Polyhedra_Library::Row, 1561
construct_epsilon_geq_zero
Parma_Polyhedra_Library::Constraint, 573
construct_mpq_base
Parma_Polyhedra_Library::Checked, 188
construct_mpq_float
Parma_Polyhedra_Library::Checked, 188
construct_mpz_base
Parma_Polyhedra_Library::Checked, 188
construct_mpz_float
Parma_Polyhedra_Library::Checked, 188
construct_upward_approximation
Parma_Polyhedra_Library::DB_Row, 627
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 893
contains
Parma_Polyhedra_Library::BD_Shape, 291
Parma_Polyhedra_Library::Box, 396
Parma_Polyhedra_Library::Grid, 789
Parma_Polyhedra_Library::Interval, 910
Parma_Polyhedra_Library::Octagonal_Shape, 1179
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1267
Parma_Polyhedra_Library::Pointset_-Powerset, 1374
Parma_Polyhedra_Library::Polyhedron, 1458
contains_integer_point
Parma_Polyhedra_Library::BD_Shape, 292
Parma_Polyhedra_Library::Box, 396
Parma_Polyhedra_Library::Grid, 789
Parma_Polyhedra_Library::Interval, 910
Parma_Polyhedra_Library::Octagonal_Shape, 1179
Parma_Polyhedra_Library::Pointset_-Powerset, 1374
Parma_Polyhedra_Library::Polyhedron, 1459
contains_restriction
Parma_Polyhedra_Library, 133
CONTROL_PARAMETER_NAME_SIZE
Parma_Polyhedra_Library::PIP_Problem, 1308
CONTROL_PARAMETER_VALUE_SIZE
Parma_Polyhedra_Library::PIP_Problem, 1308
Control_Parameter_Name
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::PIP_Problem, 1308
Control_Parameter_Value
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::PIP_Problem, 1308
control_parameters
Parma_Polyhedra_Library::PIP_Problem, 1319
control_parameters_copy
Parma_Polyhedra_Library::PIP_Problem, 1313
control_parameters_init
Parma_Polyhedra_Library::PIP_Problem, 1313
conversion
Parma_Polyhedra_Library::Grid, 789

Parma_Polyhedra_Library::Polyhedron, 1459
conversion.cc, 1764
copy_construct
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 893
copy_construct_coefficients
Parma_Polyhedra_Library::DB_Row, 628
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 893
Parma_Polyhedra_Library::Row, 1562
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 885
copy_generic
Parma_Polyhedra_Library::Checked, 189
copy_mpz
Parma_Polyhedra_Library::Checked, 189
count
Parma_Polyhedra_Library::Init, 900
count_ones
Parma_Polyhedra_Library::Bit_Row, 358
create
Parma_Polyhedra_Library::Congruence, 493
CS_PENDING
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
Status, 1629
csp
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 543
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 538
current_output_function
Parma_Polyhedra_Library::Variable, 1671
current_solution
Parma_Polyhedra_Library::PIP_Problem, 1319
CUTTING_STRATEGY
Parma_Polyhedra_Library::PIP_Problem, 1308
CUTTING_STRATEGY_ALL
Parma_Polyhedra_Library::PIP_Problem, 1308
CUTTING_STRATEGY_DEEPEST
Parma_Polyhedra_Library::PIP_Problem, 1308
CUTTING_STRATEGY_FIRST
Parma_Polyhedra_Library::PIP_Problem, 1308

d1
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1287
d2
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1288
DB_Matrix
Parma_Polyhedra_Library::DB_Matrix, 611, 616
DB_Matrix.defs.hh, 1764
DB_Matrix.inlines.hh, 1766
DB_Matrix.templates.hh, 1767
DB_Matrix.types.hh, 1768
DB_Row
Parma_Polyhedra_Library::DB_Row, 624, 625
DB_Row< T >
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 897
DB_Row.defs.hh, 1768
PPL_DB_ROW_EXTRA_DEBUG, 1770
DB_Row.inlines.hh, 1770
DB_Row.templates.hh, 1771
DB_Row.types.hh, 1772
DB_Row_Impl_Handler
Parma_Polyhedra_Library::DB_Row_Impl_-Handler, 634
dbm
Parma_Polyhedra_Library::BD_Shape, 336
DEBUG_ROUND_NOT_NEEDED
Checked_Number.inlines.hh, 1746
dec
Parma_Polyhedra_Library::float_ibm_single, 672
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 686
Parma_Polyhedra_Library::float_ieee754_-single, 691
Parma_Polyhedra_Library::float_intel_-double_extended, 696
deduce_minus_v_pm_u_bounds
Parma_Polyhedra_Library::Octagonal_Shape, 1180
deduce_u_minus_v_bounds
Parma_Polyhedra_Library::BD_Shape, 292
deduce_v_minus_u_bounds
Parma_Polyhedra_Library::BD_Shape, 292
deduce_v_pm_u_bounds
Parma_Polyhedra_Library::Octagonal_Shape, 1180
default_output_function
Parma_Polyhedra_Library::Variable, 1667
default_stop_points

Parma_Polyhedra_Library::Octagonal_Shape, 1225
default_value
Parma_Polyhedra_Library::Interval_-NS::Property, 1536
definitely_entails
Parma_Polyhedra_Library::Determinate, 639
Parma_Polyhedra_Library::Powerset, 1525
definitions.dox, 1772
Degenerate_Element
PPL_CXX_interface, 102
del_reference
Parma_Polyhedra_Library::Determinate::Rep, 1544
Delta
Parma_Polyhedra_Library::Weightwatch_-Traits, 1677
denom
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1658
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 258
denominator
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1655
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 256
derived_object
Parma_Polyhedra_Library::Is_Same_Or_-Derived, 983
Det_PSET
Parma_Polyhedra_Library::Pointset_-Powerset, 1358
Determinate
Parma_Polyhedra_Library::Determinate, 638
Determinate.defs.hh, 1772
Determinate.inlines.hh, 1774
diff_restriction
Parma_Polyhedra_Library, 133, 134
difference_assign
Parma_Polyhedra_Library::BD_Shape, 293
Parma_Polyhedra_Library::Box, 396
Parma_Polyhedra_Library::Grid, 790
Parma_Polyhedra_Library::Interval, 910, 911
Parma_Polyhedra_Library::Octagonal_Shape, 1181
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1268
Parma_Polyhedra_Library::Pointset_-Powerset, 1375
Parma_Polyhedra_Library::Polyhedron, 1462
difference_type
Parma_Polyhedra_Library::const_iterator_-to_const, 556
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 531
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Matrix::const_-iterator, 526
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 244
dim_kinds
Parma_Polyhedra_Library::Grid, 830
Dimension_Kind
Parma_Polyhedra_Library::Grid, 766
Dimension_Kinds
Parma_Polyhedra_Library::Grid, 765
dimension_type
PPL_CXX_interface, 100
Direct_Product
Parma_Polyhedra_Library::Domain_Product, 645
distances.defs.hh, 1774
distances.inlines.hh, 1775
distances.types.hh, 1776
div_2exp_assign
Parma_Polyhedra_Library, 134
Parma_Polyhedra_Library::Checked_Number, 467
Parma_Polyhedra_Library::GMP_Integer, 747
div_2exp_float
Parma_Polyhedra_Library::Checked, 189
div_2exp_mpq
Parma_Polyhedra_Library::Checked, 189
div_2exp_mpz
Parma_Polyhedra_Library::Checked, 189
div_2exp_signed_int
Parma_Polyhedra_Library::Checked, 189
div_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 190
div_assign
Parma_Polyhedra_Library::Interval, 911
div_float
Parma_Polyhedra_Library::Checked, 190
div_mpq
Parma_Polyhedra_Library::Checked, 190
div_mpz
Parma_Polyhedra_Library::Checked, 190
div_restriction
Parma_Polyhedra_Library, 134
div_round_up
Parma_Polyhedra_Library, 134
div_signed_int
Parma_Polyhedra_Library::Checked, 190
div_unsigned_int
Parma_Polyhedra_Library::Checked, 191
divisor

Parma_Polyhedra_Library::Generator, 712
Parma_Polyhedra_Library::Grid_Generator, 846
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 949
Parma_Polyhedra_Library::Simple_-Restriction_Integer_Modulo, 1596
Domain1
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1254
domain1
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1268
Domain2
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1254
domain2
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1268
drop_disjunct
Parma_Polyhedra_Library::Powerset, 1525
drop_disjuncts
Parma_Polyhedra_Library::Powerset, 1525
drop_some_non_integer_points
Parma_Polyhedra_Library::BD_Shape, 293, 294
Parma_Polyhedra_Library::Box, 397
Parma_Polyhedra_Library::Grid, 790, 791
Parma_Polyhedra_Library::Interval, 911
Parma_Polyhedra_Library::Octagonal_Shape, 1181, 1182
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1268, 1269
Parma_Polyhedra_Library::Pointset_-Powerset, 1376
Parma_Polyhedra_Library::Polyhedron, 1463, 1464
drop_some_non_integer_points_helper
Parma_Polyhedra_Library::BD_Shape, 294
Parma_Polyhedra_Library::Octagonal_Shape, 1182

e
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 250
element_begin
Parma_Polyhedra_Library::OR_Matrix, 1235
element_end
Parma_Polyhedra_Library::OR_Matrix, 1236
element_iterator
Parma_Polyhedra_Library::OR_Matrix, 1233
element_type
Parma_Polyhedra_Library::Pointset_-Powerset, 1358
EMPTY
Parma_Polyhedra_Library::BD_-Shape::Status, 1621
Parma_Polyhedra_Library::Box::Status, 1615
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1648
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
PPL_CXX_interface, 102
Status, 1629
empty
Parma_Polyhedra_Library::Bit_Row, 358
Parma_Polyhedra_Library::Congruence_-System, 513
Parma_Polyhedra_Library::Constraint_-System, 597
Parma_Polyhedra_Library::Generator_-System, 736
Parma_Polyhedra_Library::Grid_Generator_-System, 866
Parma_Polyhedra_Library::Powerset, 1526
empty_intersection_assign
Parma_Polyhedra_Library::Interval, 912
EMPTY_UP_TO_DATE
Parma_Polyhedra_Library::Box::Status, 1615
end
Parma_Polyhedra_Library::Congruence_-System, 513
Parma_Polyhedra_Library::Constraint_-System, 598
Parma_Polyhedra_Library::DB_Matrix, 613
Parma_Polyhedra_Library::DB_Row, 628
Parma_Polyhedra_Library::Generator_-System, 736
Parma_Polyhedra_Library::Grid_Generator_-System, 866
Parma_Polyhedra_Library::Matrix, 1086
Parma_Polyhedra_Library::Powerset, 1526
ENSURE_SORTEDNESS
Polyhedron_public.cc, 1896
epsilon_geq_zero
Parma_Polyhedra_Library::Constraint, 573
epsilon_geq_zero_p
Parma_Polyhedra_Library::Constraint, 587
epsilon_leq_one
Parma_Polyhedra_Library::Constraint, 574
epsilon_leq_one_p
Parma_Polyhedra_Library::Constraint, 587
eq
Parma_Polyhedra_Library::Checked, 191, 220
eq_p
Parma_Polyhedra_Library::Checked, 191
eq_restriction

Parma_Polyhedra_Library, 135
EQUAL
PPL_CXX_interface, 103
equal
Parma_Polyhedra_Library, 135
EQUALITY
Parma_Polyhedra_Library::Constraint, 571
Parma_Polyhedra_Library::Grid, 766
erase_artificials
Parma_Polyhedra_Library::MIP_Problem, 1114
erase_to_end
Parma_Polyhedra_Library::Grid_Generator_-System, 867
Parma_Polyhedra_Library::Matrix, 1086
euclidean_distance_assign
Parma_Polyhedra_Library::BD_Shape, 331
Parma_Polyhedra_Library::Box, 431
Parma_Polyhedra_Library::DB_Matrix, 616
Parma_Polyhedra_Library::Generator, 720
Parma_Polyhedra_Library::Octagonal_Shape, 1220
Parma_Polyhedra_Library::OR_Matrix, 1242
evaluate_objective_function
Parma_Polyhedra_Library::MIP_Problem, 1114
EVERYTHING
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1413
exact_div_assign
Parma_Polyhedra_Library, 135
Parma_Polyhedra_Library::Checked_Number, 467
Parma_Polyhedra_Library::GMP_Integer, 747
EXP_MASK
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-single, 693
expand_space_dimension
Parma_Polyhedra_Library::BD_Shape, 294
Parma_Polyhedra_Library::Box, 398
Parma_Polyhedra_Library::Grid, 791
Parma_Polyhedra_Library::Octagonal_Shape, 1182
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1269
Parma_Polyhedra_Library::Pointset_-Powerset, 1376
Parma_Polyhedra_Library::Polyhedron, 1464
expand_within_capacity
Parma_Polyhedra_Library::DB_Row, 628
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 894
Parma_Polyhedra_Library::Row, 1562
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 886
exponent
Parma_Polyhedra_-Library::Checked::number_struct, 1145
EXPONENT_BIAS
Parma_Polyhedra_Library::float_ibm_double, 671
Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 697
EXPONENT_BITS
Parma_Polyhedra_Library::float_ibm_double, 671
Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 697
EXPONENT_MAX
Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 697
EXPONENT_MIN

Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 698
EXPONENT_MIN_DENORM
Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_ieee754_-single, 693
Parma_Polyhedra_Library::float_intel_-double_extended, 698
external_memory_in_bytes
Parma_Polyhedra_Library, 136
Parma_Polyhedra_Library::BD_Shape, 295
Parma_Polyhedra_Library::Bit_Matrix, 349
Parma_Polyhedra_Library::Bit_Row, 358
Parma_Polyhedra_Library::Box, 398
Parma_Polyhedra_Library::Checked_Number, 467
Parma_Polyhedra_Library::Congruence, 494
Parma_Polyhedra_Library::Congruence_-System, 514
Parma_Polyhedra_Library::Constraint, 574
Parma_Polyhedra_Library::Constraint_-System, 598
Parma_Polyhedra_Library::DB_Matrix, 613
Parma_Polyhedra_Library::DB_Row, 629
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 894
Parma_Polyhedra_Library::Determinate, 639
Parma_Polyhedra_Library::Determinate::Rep, 1545
Parma_Polyhedra_Library::Generator, 712
Parma_Polyhedra_Library::Generator_-System, 737
Parma_Polyhedra_Library::Grid, 791
Parma_Polyhedra_Library::Grid_Generator, 846
Parma_Polyhedra_Library::Grid_Generator_-System, 867
Parma_Polyhedra_Library::Interval, 912
Parma_Polyhedra_Library::Linear_-Expression, 1029
Parma_Polyhedra_Library::Linear_System, 1065
Parma_Polyhedra_Library::Matrix, 1087
Parma_Polyhedra_Library::MIP_Problem, 1115
Parma_Polyhedra_Library::Octagonal_Shape, 1183
Parma_Polyhedra_Library::OR_Matrix, 1236
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1270
Parma_Polyhedra_Library::PIP_Decision_-Node, 1294
Parma_Polyhedra_Library::PIP_Problem, 1313
Parma_Polyhedra_Library::PIP_Solution_-Node, 1326
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1656
Parma_Polyhedra_Library::PIP_Tree_Node, 1341
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 256
Parma_Polyhedra_Library::Pointset_-Powerset, 1377
Parma_Polyhedra_Library::Polyhedron, 1464
Parma_Polyhedra_Library::Powerset, 1527
Parma_Polyhedra_Library::Row, 1562
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 886
Parma_Polyhedra_Library::Variable, 1667
Parma_Polyhedra_Library::Variables_Set, 1674
external_space_dim
Parma_Polyhedra_Library::MIP_Problem, 1128
Parma_Polyhedra_Library::PIP_Problem, 1319
extract_bounded_difference
Parma_Polyhedra_Library::BD_Shape, 332
extract_interval_congruence
Parma_Polyhedra_Library, 137
extract_interval_constraint
Parma_Polyhedra_Library::Box, 431
extract_octagonal_difference
Parma_Polyhedra_Library::Octagonal_Shape, 1220

f_info
Parma_Polyhedra_Library::Interval_NS, 236
f_is_empty
Parma_Polyhedra_Library, 137

Parma_Polyhedra_Library::Interval_NS, 237
f_is_singleton
Parma_Polyhedra_Library, 137
Parma_Polyhedra_Library::Interval_NS, 237
f_lower
Parma_Polyhedra_Library::Interval_NS, 237
f_OK
Parma_Polyhedra_Library, 137
f_upper
Parma_Polyhedra_Library::Interval_NS, 238
false_child
Parma_Polyhedra_Library::PIP_Decision_-Node, 1296
fdl.dox, 1777
feasible_point
Parma_Polyhedra_Library::MIP_Problem, 1115
fill_constraint_system_PR
Parma_Polyhedra_-Library::Implementation::Termination, 228
fill_constraint_system_PR_original
Parma_Polyhedra_-Library::Implementation::Termination, 229
fill_constraint_systems_MS
Parma_Polyhedra_-Library::Implementation::Termination, 230
finalize
Parma_Polyhedra_Library, 138
Parma_Polyhedra_Library::Congruence, 494
Parma_Polyhedra_Library::Congruence_-System, 514
Parma_Polyhedra_Library::Constraint, 574
Parma_Polyhedra_Library::Constraint_-System, 599
Parma_Polyhedra_Library::Euclidean_-Distance_Specialization, 648
Parma_Polyhedra_Library::Generator, 712
Parma_Polyhedra_Library::Generator_-System, 737
Parma_Polyhedra_Library::Grid_Generator, 846
Parma_Polyhedra_Library::Grid_Generator_-System, 867
Parma_Polyhedra_Library::L_Infinity_-Distance_Specialization, 997
Parma_Polyhedra_Library::Linear_-Expression, 1029
Parma_Polyhedra_Library::Polyhedron, 1465
Parma_Polyhedra_Library::Rectilinear_-Distance_Specialization, 1541
first
Parma_Polyhedra_Library::Bit_Row, 359
Parma_Polyhedra_Library::OR_-Matrix::Pseudo_Row, 1539
first_free_bit
Parma_Polyhedra_Library::Linear_-Row::Flags, 667
Parma_Polyhedra_Library::Row::Flags, 660
first_one
Parma_Polyhedra_Library::Implementation, 225
first_pending_constraint
Parma_Polyhedra_Library::MIP_Problem, 1128
Parma_Polyhedra_Library::PIP_Problem, 1319
first_pending_row
Parma_Polyhedra_Library::Linear_System, 1065
Flags
Parma_Polyhedra_Library::Linear_-Row::Flags, 663, 664
Parma_Polyhedra_Library::Row::Flags, 658
flags
Parma_Polyhedra_Library::BD_-Shape::Status, 1621
Parma_Polyhedra_Library::Box::Status, 1615
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_Library::Linear_Row, 1048
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1648
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1413
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
Parma_Polyhedra_Library::Row, 1563
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 886
Status, 1629
flags_
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 889
flags_t
Parma_Polyhedra_Library::BD_-Shape::Status, 1618
Parma_Polyhedra_Library::Box::Status, 1611
Parma_Polyhedra_Library::Grid::Status, 1603
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1646
Parma_Polyhedra_Library::Poly_Con_-Relation, 1402
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1410

Parma_Polyhedra_-
Library::Polyhedron::Status, 1634
Status, 1625
Float.cc, 1777
Float.defs.hh, 1777
PPL_NAN, 1779
Float.inlines.hh, 1779
floor_assign
Parma_Polyhedra_Library::Checked_Number, 467
floor_float
Parma_Polyhedra_Library::Checked, 191
floor_mpq
Parma_Polyhedra_Library::Checked, 191
fma
Parma_Polyhedra_Library::Checked, 191, 192
FOK
globals.defs.hh, 1788
fold_space_dimensions
Parma_Polyhedra_Library::BD_Shape, 295
Parma_Polyhedra_Library::Box, 398
Parma_Polyhedra_Library::Grid, 792
Parma_Polyhedra_Library::Octagonal_Shape, 1183
Parma_Polyhedra_Library::Partially_-
Reduced_Product, 1270
Parma_Polyhedra_Library::Pointset_-
Powerset, 1377
Parma_Polyhedra_Library::Polyhedron, 1465
forget_all_dbm_constraints
Parma_Polyhedra_Library::BD_Shape, 296
forget_all_octagonal_constraints
Parma_Polyhedra_Library::Octagonal_Shape, 1184
forget_binary_dbm_constraints
Parma_Polyhedra_Library::BD_Shape, 296
forget_binary_octagonal_constraints
Parma_Polyhedra_Library::Octagonal_Shape, 1184
fp
Parma_Polyhedra_Library::stdiobuf, 1653
fpu_direct_rounding
Parma_Polyhedra_Library::Checked, 192
fpu_inverse_rounding
Parma_Polyhedra_Library::Checked, 192
frequency
Parma_Polyhedra_Library::BD_Shape, 296
Parma_Polyhedra_Library::Box, 399
Parma_Polyhedra_Library::Grid, 792
Parma_Polyhedra_Library::Octagonal_Shape, 1184
Parma_Polyhedra_Library::Polyhedron, 1465
frequency_no_check
Parma_Polyhedra_Library::Grid, 793
From
Parma_Polyhedra_Library::Checked, 220
From1
Parma_Polyhedra_Library::Checked, 221
From2
Parma_Polyhedra_Library::Checked, 221
from_delta
Parma_Polyhedra_Library::Weightwatch_-
Traits, 1677
func
Parma_Polyhedra_Library::Is_Same_Or_-
Derived, 983
function
Parma_Polyhedra_Library::maybe_assign_-
struct, 1097
Parma_Polyhedra_Library::maybe_assign_-
struct< Type, Type >, 1098

G_MINIMIZED
Parma_Polyhedra_Library::Grid::Status, 1609
Parma_Polyhedra_-
Library::Polyhedron::Status, 1643
Status, 1629
G_UP_TO_DATE
Parma_Polyhedra_Library::Grid::Status, 1609
Parma_Polyhedra_-
Library::Polyhedron::Status, 1643
Status, 1630
gauss
Parma_Polyhedra_Library::Linear_System, 1066
gcd_assign
Parma_Polyhedra_Library, 138
Parma_Polyhedra_Library::Checked_Number, 468
Parma_Polyhedra_Library::GMP_Integer, 747
gcd_exact
Parma_Polyhedra_Library::Checked, 192
gcd_exact_noabs
Parma_Polyhedra_Library::Checked, 192
gcd_mpz
Parma_Polyhedra_Library::Checked, 193
gcdext_assign
Parma_Polyhedra_Library, 138
Parma_Polyhedra_Library::Checked_Number, 468
Parma_Polyhedra_Library::GMP_Integer, 747
gcdext_exact
Parma_Polyhedra_Library::Checked, 193
gcdext_mpz
Parma_Polyhedra_Library::Checked, 193
GEN_VIRTUAL
Parma_Polyhedra_Library::Grid, 766
gen_sys

Parma_Polyhedra_Library::Grid, 830
Parma_Polyhedra_Library::Polyhedron, 1512
generalized_affine_image
Parma_Polyhedra_Library::BD_Shape, 297, 298
Parma_Polyhedra_Library::Box, 399, 400
Parma_Polyhedra_Library::Grid, 793, 794
Parma_Polyhedra_Library::Octagonal_Shape, 1185, 1186
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1270, 1271
Parma_Polyhedra_Library::Pointset_-Powerset, 1378
Parma_Polyhedra_Library::Polyhedron, 1466
generalized_affine_preimage
Parma_Polyhedra_Library::BD_Shape, 299
Parma_Polyhedra_Library::Box, 401
Parma_Polyhedra_Library::Grid, 795
Parma_Polyhedra_Library::Octagonal_Shape, 1186, 1187
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1271, 1272
Parma_Polyhedra_Library::Pointset_-Powerset, 1379
Parma_Polyhedra_Library::Polyhedron, 1467, 1468
generate_cut
Parma_Polyhedra_Library::PIP_Solution_-Node, 1327
Generator
Parma_Polyhedra_Library::Generator, 709, 710
Generator.cc, 1779
Generator.defs.hh, 1780
Generator.inlines.hh, 1781
Generator.types.hh, 1782
Generator_System
Parma_Polyhedra_Library::Generator_-System, 732, 733
Parma_Polyhedra_Library::Generator_-System::const_iterator, 548
Generator_System.cc, 1782
Generator_System.defs.hh, 1782
Generator_System.inlines.hh, 1784
Generator_System.types.hh, 1784
generator_widening_assign
Parma_Polyhedra_Library::Grid, 796
generators
Parma_Polyhedra_Library::Polyhedron, 1468
generators_are_minimized
Parma_Polyhedra_Library::Grid, 796
Parma_Polyhedra_Library::Polyhedron, 1469
generators_are_up_to_date
Parma_Polyhedra_Library::Grid, 797
Parma_Polyhedra_Library::Polyhedron, 1469
geometrically_covers
Parma_Polyhedra_Library::Pointset_-Powerset, 1380
geometrically_equals
Parma_Polyhedra_Library::Pointset_-Powerset, 1381
get
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_-If< Is_Native_Or_Checked< From >::value >::type >, 1547
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, type-name From::info_type >::value >::type >, 1548
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_-Base, typename From::info_type >::value >::type >, 1549
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, type-name From::info_type >::value >::type >, 1550
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_-Checked< From >::value >::type >, 1551
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, type-name Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, type-name From::info_type >::value >::type >, 1552
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_-Derived< Interval_Restriction_Integer_-Modulo_Base, typename From::info_type >::value >::type >, 1553
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, type-name Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, type-name From::info_type >::value >::type >, 1554
Parma_Polyhedra_Library::Weightwatch_-

Traits, 1677
get_big_parameter_dimension
Parma_Polyhedra_Library::PIP_Problem, 1314
get_bit
Parma_Polyhedra_Library::Interval_NS, 238
get_bits
Parma_Polyhedra_Library::Interval_NS, 238
Parma_Polyhedra_Library::Row::Flags, 658
get_boundary_property
Parma_Polyhedra_Library::Interval_Info_-Bitset, 935
Parma_Polyhedra_Library::Interval_Info_-Null, 939
Parma_Polyhedra_Library::Interval_Info_-Null_Open, 942
get_control_parameter
Parma_Polyhedra_Library::MIP_Problem, 1115
Parma_Polyhedra_Library::PIP_Problem, 1314
get_digit
Parma_Polyhedra_Library::Checked, 193
get_exiting_base_index
Parma_Polyhedra_Library::MIP_Problem, 1115
get_field
Parma_Polyhedra_-Library::Polyhedron::Status, 1642
get_flags
Parma_Polyhedra_Library::Poly_Con_-Relation, 1402
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1410
get_integer
Parma_Polyhedra_Library::Interval_-Restriction_Integer, 944
Parma_Polyhedra_Library::Simple_-Restriction_Integer, 1595
get_interval
Parma_Polyhedra_Library::Box, 402
get_interval_property
Parma_Polyhedra_Library::Interval_Info_-Bitset, 935
Parma_Polyhedra_Library::Interval_Info_-Null, 940
get_limiting_box
Parma_Polyhedra_Library::Box, 402
get_limiting_octagon
Parma_Polyhedra_Library::Octagonal_Shape, 1188
get_limiting_shape
Parma_Polyhedra_Library::BD_Shape, 300
get_lower_bound
Parma_Polyhedra_Library::Box, 402
get_mp_size
Parma_Polyhedra_Library::Checked, 193
get_output_function
Parma_Polyhedra_Library::Variable, 1667
get_owner
Parma_Polyhedra_Library::PIP_Tree_Node, 1341
get_upper_bound
Parma_Polyhedra_Library::Box, 403
globals.cc, 1785
globals.defs.hh, 1785
FOK, 1788
PPL_DIRTY_TEMP_COEFFICIENT, 1788
PPL_OUTPUT_2_PARAM_TEMPLATE_-DEFINITIONS, 1790
PPL_OUTPUT_3_PARAM_TEMPLATE_-DEFINITIONS, 1790
PPL_OUTPUT_DECLARATIONS, 1790
PPL_OUTPUT_DEFINITIONS, 1790
PPL_OUTPUT_DEFINITIONS_ASCII_-ONLY, 1790
PPL_OUTPUT_TEMPLATE_-DEFINITIONS, 1790
PPL_OUTPUT_TEMPLATE_-DEFINITIONS_ASCII_ONLY, 1790
PPL_PROFILE_ADD_WEIGHT, 1791
PPL_STR, 1791
PPL_XSTR, 1791
WEIGHT_ADD, 1791
WEIGHT_ADD_MUL, 1791
WEIGHT_BEGIN, 1792
globals.inlines.hh, 1792
globals.types.hh, 1793
GMP_Integer
Parma_Polyhedra_Library, 122
GMP_Integer.defs.hh, 1795
GMP_Integer.inlines.hh, 1796
GMP_Integer.types.hh, 1796
gpl.dox, 1797
GREATER_OR_EQUAL
PPL_CXX_interface, 103
GREATER_THAN
PPL_CXX_interface, 103
greater_or_equal
Parma_Polyhedra_Library, 138
greater_than
Parma_Polyhedra_Library, 139
Grid
Parma_Polyhedra_Library::Congruence_-System, 521
Parma_Polyhedra_Library::Grid, 766–771
Parma_Polyhedra_Library::Grid_Generator, 855

Parma_Polyhedra_Library::Grid_Generator_-System, 875
Grid.defs.hh, 1797
PPL_IN_Grid_CLASS, 1799
Grid.inlines.hh, 1799
Grid.templates.hh, 1800
Grid.types.hh, 1800
Grid_Certificate
Parma_Polyhedra_Library::Congruence_-System, 522
Parma_Polyhedra_Library::Grid_Certificate, 833
Grid_Certificate.cc, 1801
Grid_Certificate.defs.hh, 1801
Grid_Certificate.inlines.hh, 1802
Grid_Certificate.types.hh, 1803
Grid_chdims.cc, 1803
Grid_conversion.cc, 1803
Grid_Generator
Parma_Polyhedra_Library::Grid_Generator, 844
Grid_Generator.cc, 1804
Grid_Generator.defs.hh, 1804
Grid_Generator.inlines.hh, 1805
Grid_Generator.types.hh, 1806
Grid_Generator_System
Parma_Polyhedra_Library::Grid_Generator, 855
Parma_Polyhedra_Library::Grid_Generator_-System, 863, 864
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 554
Grid_Generator_System.cc, 1806
Grid_Generator_System.defs.hh, 1806
Grid_Generator_System.inlines.hh, 1807
Grid_Generator_System.types.hh, 1808
Grid_Generator_System::const_iterator
Parma_Polyhedra_Library::Grid_Generator, 855
grid_generators
Parma_Polyhedra_Library::Grid, 797
Parma_Polyhedra_Library::Polyhedron, 1469
grid_line
Parma_Polyhedra_Library::Grid_Generator, 847
Grid_nonpublic.cc, 1809
BE_LAZY, 1809
grid_point
Parma_Polyhedra_Library::Grid_Generator, 847
Grid_public.cc, 1809
Grid_simplify.cc, 1810
Grid_widenings.cc, 1810
grow
Parma_Polyhedra_Library::DB_Matrix, 613
Parma_Polyhedra_Library::OR_Matrix, 1236
GS_PENDING
Parma_Polyhedra_Library::Grid::Status, 1609
Parma_Polyhedra_-Library::Polyhedron::Status, 1643
Status, 1630
gsp
Parma_Polyhedra_Library::Generator_-System::const_iterator, 549

H79_Certificate
Parma_Polyhedra_Library::H79_Certificate, 877, 878
H79_Certificate.cc, 1811
H79_Certificate.defs.hh, 1811
H79_Certificate.inlines.hh, 1812
H79_Certificate.types.hh, 1813
H79_widening_assign
Parma_Polyhedra_Library::BD_Shape, 300
Parma_Polyhedra_Library::Polyhedron, 1470
handle_result
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
Parma_Polyhedra_Library::Extended_-Number_Policy, 652
has_a_free_dimension
Parma_Polyhedra_Library::Congruence_-System, 514
has_closure_points
Parma_Polyhedra_Library::Generator_-System, 737
has_equalities
Parma_Polyhedra_Library::Constraint_-System, 599
has_linear_equalities
Parma_Polyhedra_Library::Congruence_-System, 514
has_no_rows
Parma_Polyhedra_Library::Matrix, 1087
has_nontrivial_weakening
Parma_Polyhedra_Library::Determinate, 640
has_pending_constraints
Parma_Polyhedra_Library::Polyhedron, 1470
has_pending_generators
Parma_Polyhedra_Library::Polyhedron, 1471
has_points
Parma_Polyhedra_Library::Generator_-System, 737
Parma_Polyhedra_Library::Grid_Generator_-System, 867
has_restriction
Parma_Polyhedra_Library::Interval, 912

Parma_Polyhedra_Library::Interval_-Restriction_Integer, 944
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 948
Parma_Polyhedra_Library::Interval_-Restriction_None_Base, 951
has_something_pending
Parma_Polyhedra_Library::Polyhedron, 1471
has_strict_inequalities
Parma_Polyhedra_Library::Constraint_-System, 599
hash_code
Parma_Polyhedra_Library::BD_Shape, 301
Parma_Polyhedra_Library::Grid, 797
Parma_Polyhedra_Library::Octagonal_Shape, 1188
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1272
Parma_Polyhedra_Library::Pointset_-Powerset, 1381
Parma_Polyhedra_Library::Polyhedron, 1471
homogeneous_assign
Parma_Polyhedra_Library::Scalar_Products, 1580, 1581
homogeneous_sign
Parma_Polyhedra_Library::Scalar_Products, 1582

i
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 543
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 538
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 534
Parma_Polyhedra_Library::Generator_-System::const_iterator, 549
Parma_Polyhedra_Library::Matrix::const_-iterator, 529
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 250
i_variables
Parma_Polyhedra_Library::MIP_Problem, 1128
i_vars
Parma_Polyhedra_Library::MIP_-Problem::RAII_Temporary_Real_-Relaxation, 1540
id
Parma_Polyhedra_Library::Variable, 1667
idiv_float
Parma_Polyhedra_Library::Checked, 193
idiv_mpq
Parma_Polyhedra_Library::Checked, 194
idiv_mpz
Parma_Polyhedra_Library::Checked, 194
idiv_signed_int
Parma_Polyhedra_Library::Checked, 194
idiv_unsigned_int
Parma_Polyhedra_Library::Checked, 194
Impl
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 892, 893
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 885
impl
Parma_Polyhedra_Library::DB_Row_Impl_-Handler, 634
Parma_Polyhedra_Library::Row_Impl_-Handler, 1570
implies
Parma_Polyhedra_Library::Poly_Con_-Relation, 1403
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1410, 1411
in_assert
Parma_Polyhedra_Library::Implementation, 226
inc
Parma_Polyhedra_Library::float_ibm_single, 672
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 686
Parma_Polyhedra_Library::float_ieee754_-single, 691
Parma_Polyhedra_Library::float_intel_-double_extended, 696
increase_space_dimension
Parma_Polyhedra_Library::Congruence_-System, 515
incremental_shortest_path_closure_assign
Parma_Polyhedra_Library::BD_Shape, 301
incremental_strong_closure_assign
Parma_Polyhedra_Library::Octagonal_Shape, 1188
indent_and_print
Parma_Polyhedra_Library::PIP_Tree_Node, 1342
index
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 246
index_first_pending
Parma_Polyhedra_Library::Linear_System, 1078

info
Parma_Polyhedra_Library::Interval, 912, 913
info_type
Parma_Polyhedra_Library::Interval, 906
inhomogeneous_term
Parma_Polyhedra_Library::Congruence, 494
Parma_Polyhedra_Library::Constraint, 574
Parma_Polyhedra_Library::Linear_-Expression, 1029
Parma_Polyhedra_Library::Linear_Row, 1048
Init
Parma_Polyhedra_Library::Init, 899
Parma_Polyhedra_Library::Variable, 1670
Init.cc, 1813
ppl_set_GMP_memory_allocation_functions, 1815
Init.defs.hh, 1815
Init.inlines.hh, 1816
Init.types.hh, 1817
initial_context
Parma_Polyhedra_Library::PIP_Problem, 1320
initialize
Parma_Polyhedra_Library, 139
Parma_Polyhedra_Library::Congruence, 494
Parma_Polyhedra_Library::Congruence_-System, 515
Parma_Polyhedra_Library::Constraint, 575
Parma_Polyhedra_Library::Constraint_-System, 599
Parma_Polyhedra_Library::Generator, 713
Parma_Polyhedra_Library::Generator_-System, 738
Parma_Polyhedra_Library::Grid_Generator, 847
Parma_Polyhedra_Library::Grid_Generator_-System, 867
Parma_Polyhedra_Library::Linear_-Expression, 1030
Parma_Polyhedra_Library::Polyhedron, 1471
initialized
Parma_Polyhedra_Library::MIP_Problem, 1128
initializer.hh, 1817
input
Parma_Polyhedra_Library::Checked_Number, 468
input_cs
Parma_Polyhedra_Library::MIP_Problem, 1129
Parma_Polyhedra_Library::PIP_Problem, 1320
input_generic
Parma_Polyhedra_Library::Checked, 194
input_mpq
Parma_Polyhedra_Library::Checked, 195
input_obj_function
Parma_Polyhedra_Library::MIP_Problem, 1129
insert
Parma_Polyhedra_Library::Congruence_-System, 515, 516
Parma_Polyhedra_Library::Constraint_-System, 600
Parma_Polyhedra_Library::Generator_-System, 738
Parma_Polyhedra_Library::Grid_Generator_-System, 868
Parma_Polyhedra_Library::Linear_System, 1066
Parma_Polyhedra_Library::Variables_Set, 1674
insert_pending
Parma_Polyhedra_Library::Constraint_-System, 600
Parma_Polyhedra_Library::Generator_-System, 738
Parma_Polyhedra_Library::Linear_System, 1066
insert_verbatim
Parma_Polyhedra_Library::Congruence_-System, 516
int
Parma_Polyhedra_Library::Checked, 221
int_fast16_t
Parma_Polyhedra_Library::Checked, 177
int_fast32_t
Parma_Polyhedra_Library::Checked, 177
int_fast64_t
Parma_Polyhedra_Library::Checked, 177
int_type
Parma_Polyhedra_Library::stdiobuf, 1650
integer
Parma_Polyhedra_Library::Simple_-Restriction_Integer, 1595
integer_space_dimensions
Parma_Polyhedra_Library::MIP_Problem, 1116
integer_upper_bound_assign_if_exact
Parma_Polyhedra_Library::BD_Shape, 302
Parma_Polyhedra_Library::Octagonal_Shape, 1189
internal_space_dim
Parma_Polyhedra_Library::MIP_Problem, 1129
Parma_Polyhedra_Library::PIP_Problem, 1320
intersect_assign

Parma_Polyhedra_Library::Interval, 913, 914
intersect_restriction
Parma_Polyhedra_Library, 139
intersection_assign
Parma_Polyhedra_Library::BD_Shape, 302
Parma_Polyhedra_Library::Box, 403
Parma_Polyhedra_Library::Grid, 798
Parma_Polyhedra_Library::Octagonal_Shape, 1189
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1272
Parma_Polyhedra_Library::Pointset_-Powerset, 1382
Parma_Polyhedra_Library::Polyhedron, 1472
intersection_preserving_enlarge_element
Parma_Polyhedra_Library::Pointset_-Powerset, 1382
Interval
Parma_Polyhedra_Library::Interval, 906
Interval.defs.hh, 1818
Interval.inlines.hh, 1820
Interval.types.hh, 1823
Interval_Info.defs.hh, 1823
Interval_Info.inlines.hh, 1825
Interval_Info.types.hh, 1825
Interval_Info_Bitset
Parma_Polyhedra_Library::Interval_Info_-Bitset, 932
Interval_Info_Null_Open
Parma_Polyhedra_Library::Interval_Info_-Null_Open, 942
interval_relation
Parma_Polyhedra_Library, 139
Interval_Restriction.defs.hh, 1826
Interval_Restriction_Integer
Parma_Polyhedra_Library::Interval_-Restriction_Integer, 944
Interval_Restriction_Integer_Modulo
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 947
Interval_Restriction_None
Parma_Polyhedra_Library::Interval_-Restriction_None, 950
interval_type
Parma_Polyhedra_Library::Box, 378
IO_Operators::operator<<
Parma_Polyhedra_Library::Grid_Generator, 855
irrational_precision
Parma_Polyhedra_Library, 140
Parma_Polyhedra_Library::Checked, 221
is_additive_inverse
Parma_Polyhedra_Library, 140
is_always_topologically_closed
Parma_Polyhedra_Library::Interval, 914
is_better_pivot
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1656
is_bottom
Parma_Polyhedra_Library::Determinate, 640
Parma_Polyhedra_Library::Powerset, 1527
is_bounded
Parma_Polyhedra_Library::BD_Shape, 302
Parma_Polyhedra_Library::Box, 404
Parma_Polyhedra_Library::Grid, 798
Parma_Polyhedra_Library::Interval, 914
Parma_Polyhedra_Library::Octagonal_Shape, 1190
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1273
Parma_Polyhedra_Library::Pointset_-Powerset, 1382
Parma_Polyhedra_Library::Polyhedron, 1472
is_canonical
Parma_Polyhedra_Library, 141
is_cert_multiset_stabilizing
Parma_Polyhedra_Library::Pointset_-Powerset, 1382
is_closure_point
Parma_Polyhedra_Library::Generator, 713
is_definitely_equivalent_to
Parma_Polyhedra_Library::Determinate, 640
is_discrete
Parma_Polyhedra_Library::BD_Shape, 303
Parma_Polyhedra_Library::Box, 404
Parma_Polyhedra_Library::Grid, 798
Parma_Polyhedra_Library::Octagonal_Shape, 1190
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1273
Parma_Polyhedra_Library::Pointset_-Powerset, 1383
Parma_Polyhedra_Library::Polyhedron, 1472
IS_DISJOINT
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
is_disjoint
Parma_Polyhedra_Library::Poly_Con_-Relation, 1403
is_disjoint_from
Parma_Polyhedra_Library::BD_Shape, 303
Parma_Polyhedra_Library::Box, 404
Parma_Polyhedra_Library::Grid, 798
Parma_Polyhedra_Library::Interval, 914
Parma_Polyhedra_Library::Octagonal_Shape, 1190
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1273

Parma_Polyhedra_Library::Pointset_-Powerset, 1383
Parma_Polyhedra_Library::Polyhedron, 1472
is_empty
Parma_Polyhedra_Library::BD_Shape, 303
Parma_Polyhedra_Library::Box, 404
Parma_Polyhedra_Library::Grid, 799
Parma_Polyhedra_Library::Interval, 915
Parma_Polyhedra_Library::Octagonal_Shape, 1191
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1274
Parma_Polyhedra_Library::Pointset_-Powerset, 1383
Parma_Polyhedra_Library::Polyhedron, 1473
is_equal_at_dimension
Parma_Polyhedra_Library::Congruence, 495
Parma_Polyhedra_Library::Grid_Generator, 847
is_equal_to
Parma_Polyhedra_Library::Congruence_-System, 517
Parma_Polyhedra_Library::Grid_Generator, 848
Parma_Polyhedra_Library::Grid_Generator_-System, 868
is_equality
Parma_Polyhedra_Library::Congruence, 495
Parma_Polyhedra_Library::Constraint, 575
is_equivalent_to
Parma_Polyhedra_Library::Constraint, 576
Parma_Polyhedra_Library::Generator, 713
Parma_Polyhedra_Library::Grid_Generator, 848
is_even
Parma_Polyhedra_Library, 141
is_in_base
Parma_Polyhedra_Library::MIP_Problem, 1116
IS_INCLUDED
Parma_Polyhedra_Library::Poly_Con_-Relation, 1407
is_included
Parma_Polyhedra_Library::Poly_Con_-Relation, 1403
is_included_in
Parma_Polyhedra_Library::Grid, 799
Parma_Polyhedra_Library::Polyhedron, 1473
is_inconsistent
Parma_Polyhedra_Library::Congruence, 495
Parma_Polyhedra_Library::Constraint, 576
is_inequality
Parma_Polyhedra_Library::Constraint, 576
is_inf
Parma_Polyhedra_Library::float_ibm_single, 672
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 686
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 696
is_inf_float
Parma_Polyhedra_Library::Checked, 195
is_infinity
Parma_Polyhedra_Library, 141
Parma_Polyhedra_Library::Checked_Number, 470
Parma_Polyhedra_Library::Interval, 915
is_int_float
Parma_Polyhedra_Library::Checked, 195
is_int_int
Parma_Polyhedra_Library::Checked, 195
is_int_mpq
Parma_Polyhedra_Library::Checked, 195
is_int_mpz
Parma_Polyhedra_Library::Checked, 196
is_integer
Parma_Polyhedra_Library, 141
Parma_Polyhedra_Library::Checked_Number, 470
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1657
is_line
Parma_Polyhedra_Library::Generator, 713
Parma_Polyhedra_Library::Grid_Generator, 848
is_line_or_equality
Parma_Polyhedra_Library::Linear_Row, 1049
Parma_Polyhedra_Library::Linear_-Row::Flags, 664
is_line_or_parameter
Parma_Polyhedra_Library::Grid_Generator, 848
is_line_or_ray
Parma_Polyhedra_Library::Generator, 714
is_lp_satisfiable
Parma_Polyhedra_Library::MIP_Problem, 1116
is_matching_closure_point
Parma_Polyhedra_Library::Generator, 714
is_minf_float
Parma_Polyhedra_Library::Checked, 196
is_minf_int

Parma_Polyhedra_Library::Checked, 196
is_minf_mpq
Parma_Polyhedra_Library::Checked, 196
is_minf_mpz
Parma_Polyhedra_Library::Checked, 196
is_minus_infinity
Parma_Polyhedra_Library, 142
Parma_Polyhedra_Library::Checked_Number, 470
is_mip_satisfiable
Parma_Polyhedra_Library::MIP_Problem, 1117
is_nan
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 696
is_nan_float
Parma_Polyhedra_Library::Checked, 196
is_nan_int
Parma_Polyhedra_Library::Checked, 196
is_nan_mpq
Parma_Polyhedra_Library::Checked, 196
is_nan_mpz
Parma_Polyhedra_Library::Checked, 197
is_necessarily_closed
Parma_Polyhedra_Library::Linear_Row, 1049
Parma_Polyhedra_Library::Linear_-Row::Flags, 665
Parma_Polyhedra_Library::Linear_System, 1067
Parma_Polyhedra_Library::Polyhedron, 1473
is_necessarily_closed_for_interfaces
Parma_Polyhedra_Library::Interfaces, 232
is_nonstrict_inequality
Parma_Polyhedra_Library::Constraint, 577
is_not_a_number
Parma_Polyhedra_Library, 142, 143
Parma_Polyhedra_Library::Checked_Number, 470
is_not_necessarily_closed
Parma_Polyhedra_Library::Linear_Row, 1049
Parma_Polyhedra_Library::Linear_-Row::Flags, 665
is_omega_reduced
Parma_Polyhedra_Library::Powerset, 1527
is_parameter
Parma_Polyhedra_Library::Grid_Generator, 849
is_parameter_or_point
Parma_Polyhedra_Library::Grid_Generator, 849
is_pinf_float
Parma_Polyhedra_Library::Checked, 197
is_pinf_int
Parma_Polyhedra_Library::Checked, 197
is_pinf_mpq
Parma_Polyhedra_Library::Checked, 197
is_pinf_mpz
Parma_Polyhedra_Library::Checked, 197
is_plus_infinity
Parma_Polyhedra_Library, 143
Parma_Polyhedra_Library::Checked_Number, 470
is_point
Parma_Polyhedra_Library::Generator, 714
Parma_Polyhedra_Library::Grid_Generator, 849
is_proper_congruence
Parma_Polyhedra_Library::Congruence, 496
is_ray
Parma_Polyhedra_Library::Generator, 715
is_ray_or_point
Parma_Polyhedra_Library::Generator, 715
is_ray_or_point_or_inequality
Parma_Polyhedra_Library::Linear_Row, 1049
Parma_Polyhedra_Library::Linear_-Row::Flags, 665
is_reduced
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1274
is_satisfiable
Parma_Polyhedra_Library::MIP_Problem, 1117
Parma_Polyhedra_Library::PIP_Problem, 1314
is_satisfied
Parma_Polyhedra_Library::MIP_Problem, 1117
is_saturated
Parma_Polyhedra_Library::MIP_Problem, 1118
is_shared
Parma_Polyhedra_Library::Determinate::Rep, 1545
is_shortest_path_reduced
Parma_Polyhedra_Library::BD_Shape, 304
is_singleton
Parma_Polyhedra_Library::Interval, 915
is_singleton_integer

Parma_Polyhedra_Library, 144
is_sorted
Parma_Polyhedra_Library::Linear_System, 1067
is_stabilizing
Parma_Polyhedra_Library::BHRZ03_-Certificate, 340
Parma_Polyhedra_Library::Grid_Certificate, 834
is_strict_inequality
Parma_Polyhedra_Library::Constraint, 577
is_strong_coherent
Parma_Polyhedra_Library::Octagonal_Shape, 1191
is_strongly_reduced
Parma_Polyhedra_Library::Octagonal_Shape, 1191
is_tautological
Parma_Polyhedra_Library::Congruence, 496
Parma_Polyhedra_Library::Constraint, 577
is_top
Parma_Polyhedra_Library::Determinate, 640
Parma_Polyhedra_Library::Powerset, 1527
is_topologically_closed
Parma_Polyhedra_Library::BD_Shape, 304
Parma_Polyhedra_Library::Box, 405
Parma_Polyhedra_Library::Grid, 799
Parma_Polyhedra_Library::Interval, 916
Parma_Polyhedra_Library::Octagonal_Shape, 1191
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1274
Parma_Polyhedra_Library::Pointset_-Powerset, 1383
Parma_Polyhedra_Library::Polyhedron, 1474
is_universe
Parma_Polyhedra_Library::BD_Shape, 304
Parma_Polyhedra_Library::Box, 405
Parma_Polyhedra_Library::Grid, 799
Parma_Polyhedra_Library::Interval, 916
Parma_Polyhedra_Library::Octagonal_Shape, 1192
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1274
Parma_Polyhedra_Library::Pointset_-Powerset, 1384
Parma_Polyhedra_Library::Polyhedron, 1474
is_zero
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 696
Parma_Polyhedra_Library::Linear_-Expression, 1030
Parma_Polyhedra_Library::Linear_Row, 1050
isqrt
Parma_Polyhedra_Library, 144
isqrtrem
Parma_Polyhedra_Library::Checked, 197
Iter
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 531
Parma_Polyhedra_Library::Matrix::const_-iterator, 526
iter_swap
Parma_Polyhedra_Library::Bit_Row, 361
Parma_Polyhedra_Library::DB_Row, 631
Parma_Polyhedra_Library::Linear_Row, 1054
Parma_Polyhedra_Library::Row, 1567
iterator
Parma_Polyhedra_Library::DB_Row, 624
Parma_Polyhedra_Library::Pointset_-Powerset, 1358
Parma_Polyhedra_Library::Powerset, 1520
iterator_category
Parma_Polyhedra_Library::const_iterator_-to_const, 556
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 531
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Matrix::const_-iterator, 526
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 244
iterator_to_const
Parma_Polyhedra_Library::iterator_to_const, 994
iterator_to_const< Container >
Parma_Polyhedra_Library::const_iterator_-to_const, 559
iterator_to_const.defs.hh, 1829
iterator_to_const.inlines.hh, 1831
iterator_to_const.types.hh, 1831

join_assign
Parma_Polyhedra_Library::Interval, 916, 917
join_restriction
Parma_Polyhedra_Library, 144, 145

Kind
Parma_Polyhedra_Library::Linear_Row, 1044

l_infinity_distance_assign
Parma_Polyhedra_Library::BD_Shape, 332, 333
Parma_Polyhedra_Library::Box, 432
Parma_Polyhedra_Library::DB_Matrix, 617
Parma_Polyhedra_Library::Generator, 721
Parma_Polyhedra_Library::Octagonal_Shape, 1221, 1222
Parma_Polyhedra_Library::OR_Matrix, 1242
l_m_distance_assign
Parma_Polyhedra_Library::Box, 433
Parma_Polyhedra_Library::DB_Matrix, 617
Parma_Polyhedra_Library::Generator, 722
Parma_Polyhedra_Library::OR_Matrix, 1243
last
Parma_Polyhedra_Library::Bit_Row, 359
last_generator
Parma_Polyhedra_Library::MIP_Problem, 1129
last_one
Parma_Polyhedra_Library::Implementation, 225
lcm_assign
Parma_Polyhedra_Library, 145
Parma_Polyhedra_Library::Checked_Number, 470
Parma_Polyhedra_Library::GMP_Integer, 747
lcm_gcd_exact
Parma_Polyhedra_Library::Checked, 198
lcm_mpz
Parma_Polyhedra_Library::Checked, 198
le
Parma_Polyhedra_Library::Checked, 198, 221
le_p
Parma_Polyhedra_Library::Checked, 198
least_upper_bound_assign
Parma_Polyhedra_Library::Powerset, 1528
less
Parma_Polyhedra_Library::Variable, 1670
LESS_OR_EQUAL
PPL_CXX_interface, 103
LESS_THAN
PPL_CXX_interface, 103
less_or_equal
Parma_Polyhedra_Library, 145
less_than
Parma_Polyhedra_Library, 145
Parma_Polyhedra_Library::Weightwatch_-Traits, 1677
lift_op_assign
Parma_Polyhedra_Library::Determinate, 640
limit_precision
Parma_Polyhedra_Library::Checked, 198
limited_BHMZ05_extrapolation_assign
Parma_Polyhedra_Library::BD_Shape, 304
Parma_Polyhedra_Library::Octagonal_Shape, 1192
limited_BHRZ03_extrapolation_assign
Parma_Polyhedra_Library::Polyhedron, 1475
limited_CC76_extrapolation_assign
Parma_Polyhedra_Library::BD_Shape, 305
Parma_Polyhedra_Library::Box, 405
Parma_Polyhedra_Library::Octagonal_Shape, 1192
limited_congruence_extrapolation_assign
Parma_Polyhedra_Library::Grid, 800
limited_extrapolation_assign
Parma_Polyhedra_Library::Grid, 800
limited_generator_extrapolation_assign
Parma_Polyhedra_Library::Grid, 800
limited_H79_extrapolation_assign
Parma_Polyhedra_Library::BD_Shape, 305
Parma_Polyhedra_Library::Polyhedron, 1475
Limited_Widening_Function
Parma_Polyhedra_Library::Limited_-Widening_Function, 1016
Limited_Widening_Method
Parma_Polyhedra_Library::Limited_-Widening_Function, 1016
limiting_cs
Parma_Polyhedra_Library::Limited_-Widening_Function, 1017
lin_space_dim
Parma_Polyhedra_Library::BHRZ03_-Certificate, 341
LINE
Parma_Polyhedra_Library::Generator, 709
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Grid_Generator, 843
line
Parma_Polyhedra_Library::Generator, 715
LINE_OR_EQUALITY
Parma_Polyhedra_Library::Linear_Row, 1045
linear_combine
Parma_Polyhedra_Library::Linear_Row, 1050
Parma_Polyhedra_Library::MIP_Problem, 1118
Linear_Expression
Parma_Polyhedra_Library::Grid_Generator, 855
Parma_Polyhedra_Library::Linear_-Expression, 1024–1027
Linear_Expression.cc, 1832
Linear_Expression.defs.hh, 1832

Linear_Expression.inlines.hh, 1833
Linear_Expression.types.hh, 1834
linear_partition
Parma_Polyhedra_Library::Pointset_-Powerset, 1397
linear_partition_aux
Parma_Polyhedra_Library::Pointset_-Powerset, 1398
Linear_Row
Parma_Polyhedra_Library::Linear_Row, 1045
Linear_Row.cc, 1835
Linear_Row.defs.hh, 1835
Linear_Row.inlines.hh, 1836
Linear_Row.types.hh, 1837
Linear_System
Parma_Polyhedra_Library::Linear_System, 1061
Linear_System.cc, 1837
Linear_System.defs.hh, 1838
Linear_System.inlines.hh, 1839
Linear_System.types.hh, 1840
low_bits_mask
Parma_Polyhedra_Library, 145
lower
Parma_Polyhedra_Library::Interval, 917
lower_
Parma_Polyhedra_Library::Interval, 929
lower_approximation_difference_assign
Parma_Polyhedra_Library::Interval, 918
lower_constraint
Parma_Polyhedra_Library::Interval, 918
lower_extend
Parma_Polyhedra_Library::Interval, 918
lower_is_boundary_infinity
Parma_Polyhedra_Library::Interval, 918
lower_is_domain_inf
Parma_Polyhedra_Library::Interval, 919
lower_is_open
Parma_Polyhedra_Library::Interval, 919
lower_normalize
Parma_Polyhedra_Library::Interval, 919
lower_triangular
Parma_Polyhedra_Library::Grid, 801
lp
Parma_Polyhedra_Library::MIP_-Problem::RAII_Temporary_Real_-Relaxation, 1540
lsp
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_intel_-double_extended, 698
LSP_DMAX
Parma_Polyhedra_Library::float_intel_-double_extended, 698
LSP_INF
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-quad, 688
Parma_Polyhedra_Library::float_intel_-double_extended, 698
LSP_MAX
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-quad, 689
LSP_NMAX
Parma_Polyhedra_Library::float_intel_-double_extended, 698
LSP_ZERO
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-quad, 689
Parma_Polyhedra_Library::float_intel_-double_extended, 698
lt
Parma_Polyhedra_Library::Checked, 199, 222
lt_p
Parma_Polyhedra_Library::Checked, 199
lw_method
Parma_Polyhedra_Library::Limited_-Widening_Function, 1017

mantissa
Parma_Polyhedra_-Library::Checked::number_struct, 1146
MANTISSA_BITS
Parma_Polyhedra_Library::float_ibm_double, 671
Parma_Polyhedra_Library::float_ibm_single, 674
Parma_Polyhedra_Library::float_ieee754_-double, 679
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-quad, 689
Parma_Polyhedra_Library::float_ieee754_-single, 694
Parma_Polyhedra_Library::float_intel_-double_extended, 699
map_space_dimensions
Parma_Polyhedra_Library::BD_Shape, 306
Parma_Polyhedra_Library::Box, 406

Parma_Polyhedra_Library::Grid, 801
Parma_Polyhedra_Library::Octagonal_Shape, 1193
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1275
Parma_Polyhedra_Library::Pointset_-Powerset, 1384
Parma_Polyhedra_Library::Polyhedron, 1476
mapping
Parma_Polyhedra_Library::MIP_Problem, 1129
Parma_Polyhedra_Library::PIP_Solution_-Node, 1331
marked_empty
Parma_Polyhedra_Library::BD_Shape, 307
Parma_Polyhedra_Library::Box, 407
Parma_Polyhedra_Library::Grid, 802
Parma_Polyhedra_Library::Octagonal_Shape, 1194
Parma_Polyhedra_Library::Polyhedron, 1477
marked_shortest_path_closed
Parma_Polyhedra_Library::BD_Shape, 308
marked_shortest_path_reduced
Parma_Polyhedra_Library::BD_Shape, 308
marked_strongly_closed
Parma_Polyhedra_Library::Octagonal_Shape, 1195
marked_zero_dim_univ
Parma_Polyhedra_Library::BD_Shape, 309
Parma_Polyhedra_Library::Octagonal_Shape, 1195
math_utilities.defs.hh, 1841
math_utilities.inlines.hh, 1842
Matrix
Parma_Polyhedra_Library::Matrix, 1082
matrix
Parma_Polyhedra_Library::Octagonal_Shape, 1225
Matrix.cc, 1844
Matrix.defs.hh, 1844
Matrix.inlines.hh, 1845
Matrix.types.hh, 1846
matrix_at
Parma_Polyhedra_Library::Octagonal_Shape, 1195, 1196
max
Parma_Polyhedra_-Library::Checked::Extended_Int, 649
max_assign
Parma_Polyhedra_Library, 146
max_min
Parma_Polyhedra_Library::BD_Shape, 309
Parma_Polyhedra_Library::Box, 407, 408
Parma_Polyhedra_Library::Grid, 803
Parma_Polyhedra_Library::Octagonal_Shape, 1196, 1197
Parma_Polyhedra_Library::Polyhedron, 1477
max_num_columns
Parma_Polyhedra_Library::DB_Matrix, 614
Parma_Polyhedra_Library::Matrix, 1088
max_num_rows
Parma_Polyhedra_Library::Bit_Matrix, 349
Parma_Polyhedra_Library::DB_Matrix, 614
Parma_Polyhedra_Library::Matrix, 1088
Parma_Polyhedra_Library::OR_Matrix, 1237
max_size
Parma_Polyhedra_Library::DB_Row, 629
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 894
Parma_Polyhedra_Library::Row, 1563
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 886
max_space_dimension
Parma_Polyhedra_Library::BD_Shape, 310
Parma_Polyhedra_Library::Box, 409
Parma_Polyhedra_Library::Congruence, 496
Parma_Polyhedra_Library::Congruence_-System, 517
Parma_Polyhedra_Library::Constraint, 578
Parma_Polyhedra_Library::Constraint_-System, 601
Parma_Polyhedra_Library::Generator, 715
Parma_Polyhedra_Library::Generator_-System, 739
Parma_Polyhedra_Library::Grid, 803
Parma_Polyhedra_Library::Grid_Generator, 849
Parma_Polyhedra_Library::Grid_Generator_-System, 868
Parma_Polyhedra_Library::Linear_-Expression, 1030
Parma_Polyhedra_Library::Linear_Row, 1050
Parma_Polyhedra_Library::Linear_System, 1068
Parma_Polyhedra_Library::MIP_Problem, 1118
Parma_Polyhedra_Library::Octagonal_Shape, 1197
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1275
Parma_Polyhedra_Library::PIP_Problem, 1314
Parma_Polyhedra_Library::Pointset_-Powerset, 1384
Parma_Polyhedra_Library::Polyhedron, 1478
Parma_Polyhedra_Library::Variable, 1668
Parma_Polyhedra_Library::Variables_Set, 1674

MAXIMIZATION
PPL_CXX_interface, 103
maximize
Parma_Polyhedra_Library::BD_Shape, 310, 311
Parma_Polyhedra_Library::Box, 409
Parma_Polyhedra_Library::Grid, 804
Parma_Polyhedra_Library::Octagonal_Shape, 1198
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1275, 1276
Parma_Polyhedra_Library::Pointset_-Powerset, 1384, 1385
Parma_Polyhedra_Library::Polyhedron, 1478, 1479
maybe_abandon
Parma_Polyhedra_Library, 146
Parma_Polyhedra_Library::Throwable, 1660
maybe_assign
Parma_Polyhedra_Library, 146
maybe_check_fpu_inexact
Parma_Polyhedra_Library, 146
maybe_reset_fpu_inexact
Parma_Polyhedra_Library, 147
meet_assign
Parma_Polyhedra_Library::Determinate, 641
Parma_Polyhedra_Library::Powerset, 1528
memory_size_type
PPL_CXX_interface, 101
merge_rows_assign
Parma_Polyhedra_Library::Linear_System, 1068
merge_split_variable
Parma_Polyhedra_Library::MIP_Problem, 1118
meta_programming.hh, 1846
PPL_COMPILE_TIME_CHECK_AUX, 1848
PPL_COMPILE_TIME_CHECK_NAME, 1848
min
Parma_Polyhedra_-Library::Checked::Extended_Int, 649
min_assign
Parma_Polyhedra_Library, 147
MINIMIZATION
PPL_CXX_interface, 103
minimize
Parma_Polyhedra_Library::BD_Shape, 311, 312
Parma_Polyhedra_Library::Box, 410
Parma_Polyhedra_Library::Grid, 805, 806
Parma_Polyhedra_Library::Octagonal_Shape, 1199
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1276, 1277
Parma_Polyhedra_Library::Pointset_-Powerset, 1386
Parma_Polyhedra_Library::Polyhedron, 1479–1481
minimize.cc, 1849
minimized_congruences
Parma_Polyhedra_Library::BD_Shape, 312
Parma_Polyhedra_Library::Box, 411
Parma_Polyhedra_Library::Grid, 806
Parma_Polyhedra_Library::Octagonal_Shape, 1200
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1277
Parma_Polyhedra_Library::Polyhedron, 1481
minimized_constraints
Parma_Polyhedra_Library::BD_Shape, 313
Parma_Polyhedra_Library::Box, 411
Parma_Polyhedra_Library::Grid, 806
Parma_Polyhedra_Library::Octagonal_Shape, 1200
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1278
Parma_Polyhedra_Library::Polyhedron, 1482
minimized_generators
Parma_Polyhedra_Library::Polyhedron, 1482
minimized_grid_generators
Parma_Polyhedra_Library::Grid, 807
Parma_Polyhedra_Library::Polyhedron, 1482
MINUS_INFINITY
Parma_Polyhedra_Library, 165
minus_infinity
Parma_Polyhedra_Library, 147
Parma_Polyhedra_-Library::Checked::Extended_Int, 649
MIP_Problem
Parma_Polyhedra_Library::MIP_Problem, 1107–1109
MIP_Problem.cc, 1849
PPL_SIMPLEX_USE_MIP_HEURISTIC, 1850
MIP_Problem.defs.hh, 1850
MIP_Problem.inlines.hh, 1851
MIP_Problem.templates.hh, 1852
MIP_Problem.types.hh, 1854
MIP_Problem_Status
PPL_CXX_interface, 102
MIXED
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
modulo_type
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 947

Parma_Polyhedra_Library::Simple_-
Restriction_Integer_Modulo, 1596
modulus
Parma_Polyhedra_Library::Congruence, 497
mp_size_field_t
Parma_Polyhedra_Library::Checked, 177
msp
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 689
Parma_Polyhedra_Library::float_intel_-
double_extended, 699
MSP_NEG_INF
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 689
Parma_Polyhedra_Library::float_intel_-
double_extended, 699
MSP_NEG_ZERO
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 689
Parma_Polyhedra_Library::float_intel_-
double_extended, 699
MSP_POS_INF
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 690
Parma_Polyhedra_Library::float_intel_-
double_extended, 699
MSP_POS_ZERO
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 690
Parma_Polyhedra_Library::float_intel_-
double_extended, 699
MSP_SGN_MASK
Parma_Polyhedra_Library::float_ieee754_-
double, 680
Parma_Polyhedra_Library::float_ieee754_-
quad, 690
Parma_Polyhedra_Library::float_intel_-
double_extended, 700
mul_2exp_assign
Parma_Polyhedra_Library, 147
Parma_Polyhedra_Library::Checked_Number,
471
Parma_Polyhedra_Library::GMP_Integer, 748
mul_2exp_float
Parma_Polyhedra_Library::Checked, 199
mul_2exp_mpq
Parma_Polyhedra_Library::Checked, 199
mul_2exp_mpz
Parma_Polyhedra_Library::Checked, 199
mul_2exp_signed_int
Parma_Polyhedra_Library::Checked, 199
mul_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 200
mul_assign
Parma_Polyhedra_Library::Interval, 920
mul_float
Parma_Polyhedra_Library::Checked, 200
mul_int_larger
Parma_Polyhedra_Library::Checked, 200
mul_mpq
Parma_Polyhedra_Library::Checked, 200
mul_mpz
Parma_Polyhedra_Library::Checked, 200
mul_restriction
Parma_Polyhedra_Library, 147, 148
mul_signed_int
Parma_Polyhedra_Library::Checked, 200
mul_unsigned_int
Parma_Polyhedra_Library::Checked, 201
mulmod
Parma_Polyhedra_Library, 148
multiply_grid
Parma_Polyhedra_Library::Grid, 807
mutate
Parma_Polyhedra_Library::Determinate, 641

N
Parma_Polyhedra_Library::BD_Shape, 272
Parma_Polyhedra_Library::Octagonal_Shape,
1160
namespaces.hh, 1854
NECESSARILY_CLOSED
PPL_CXX_interface, 104
neg_assign
Parma_Polyhedra_Library, 148
Parma_Polyhedra_Library::Checked_Number,
471
Parma_Polyhedra_Library::GMP_Integer, 748
Parma_Polyhedra_Library::Interval, 920
neg_exponent
Parma_Polyhedra_-
Library::Checked::number_struct, 1146
neg_float
Parma_Polyhedra_Library::Checked, 201
NEG_INF
Parma_Polyhedra_Library::float_ibm_single,
674

Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-single, 694
neg_int_larger
Parma_Polyhedra_Library::Checked, 201
neg_mantissa
Parma_Polyhedra_-Library::Checked::number_struct, 1146
neg_mpq
Parma_Polyhedra_Library::Checked, 201
neg_mpz
Parma_Polyhedra_Library::Checked, 201
neg_restriction
Parma_Polyhedra_Library, 149, 150
neg_signed_int
Parma_Polyhedra_Library::Checked, 201
neg_unsigned_int
Parma_Polyhedra_Library::Checked, 202
NEG_ZERO
Parma_Polyhedra_Library::float_ibm_single, 675
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-single, 694
negate
Parma_Polyhedra_Library::Congruence, 497
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 677
Parma_Polyhedra_Library::float_ieee754_-half, 682
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 696
Parma_Polyhedra_Library::Grid_Generator, 849
NEGATIVE
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
new_reference
Parma_Polyhedra_Library::Determinate::Rep, 1545
next
Parma_Polyhedra_Library::Bit_Row, 359
nnc_bit
Parma_Polyhedra_Library::Linear_-Row::Flags, 667
NNC_Polyhedron
Parma_Polyhedra_Library::NNC_Polyhedron, 1137–1141
NNC_Polyhedron.cc, 1855
USE_BHZ09, 1856
NNC_Polyhedron.defs.hh, 1856
NNC_Polyhedron.inlines.hh, 1857
NNC_Polyhedron.types.hh, 1858
nnc_validity_bit
Parma_Polyhedra_Library::Linear_-Row::Flags, 668
No_Reduction
Parma_Polyhedra_Library::No_Reduction, 1144
non_redundant_matrix_entries
Parma_Polyhedra_Library::Octagonal_Shape, 1200
nonconst
checked.defs.hh, 1710
Parma_Polyhedra_Library::Checked, 222
NONE
Parma_Polyhedra_Library::Box::Status, 1615
NONSTRICT_INEQUALITY
Parma_Polyhedra_Library::Constraint, 571
normalize
Parma_Polyhedra_Library::Congruence, 497
Parma_Polyhedra_Library::Interval, 920
Parma_Polyhedra_Library::Interval_-Restriction_Integer, 945
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 948
Parma_Polyhedra_Library::Interval_-Restriction_None_Base, 951
Parma_Polyhedra_Library::Linear_System, 1068
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1657
Parma_Polyhedra_Library::Row, 1563
normalize2
Parma_Polyhedra_Library, 150
normalize_divisors
Parma_Polyhedra_Library::Grid, 807, 808
normalize_moduli
Parma_Polyhedra_Library::Congruence_-System, 517
NOT_EQUAL
PPL_CXX_interface, 103
NOT_NECESSARILY_CLOSED
PPL_CXX_interface, 104
not_a_dimension
Parma_Polyhedra_Library, 150
NOT_A_NUMBER
Parma_Polyhedra_Library, 165
not_a_number
Parma_Polyhedra_Library, 150

Parma_Polyhedra_-
Library::Checked::Extended_Int, 649
not_equal
Parma_Polyhedra_Library, 151
NOTHING
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1407
Parma_Polyhedra_Library::Poly_Gen_-
Relation, 1413
nothing
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1404
Parma_Polyhedra_Library::Poly_Gen_-
Relation, 1411
num_bits
Parma_Polyhedra_Library::Row::Flags, 661
num_columns
Parma_Polyhedra_Library::Bit_Matrix, 349
Parma_Polyhedra_Library::Grid_Generator_-
System, 869
Parma_Polyhedra_Library::Matrix, 1088
num_constraints
Parma_Polyhedra_Library::BHRZ03_-
Certificate, 341
Parma_Polyhedra_Library::H79_Certificate,
879
num_equalities
Parma_Polyhedra_Library::Congruence_-
System, 517
Parma_Polyhedra_Library::Constraint_-
System, 601
Parma_Polyhedra_Library::Grid_Certificate,
835
num_inequalities
Parma_Polyhedra_Library::Constraint_-
System, 601
num_lines
Parma_Polyhedra_Library::Generator_-
System, 739
Parma_Polyhedra_Library::Grid_Generator_-
System, 869
num_lines_or_equalities
Parma_Polyhedra_Library::Linear_System,
1069
num_parameters
Parma_Polyhedra_Library::Grid_Generator_-
System, 869
num_pending_rows
Parma_Polyhedra_Library::Linear_System,
1069
num_points
Parma_Polyhedra_Library::BHRZ03_-
Certificate, 342
num_proper_congruences
Parma_Polyhedra_Library::Congruence_-
System, 518
Parma_Polyhedra_Library::Grid_Certificate,
835
num_rays
Parma_Polyhedra_Library::Generator_-
System, 739
num_rays_null_coord
Parma_Polyhedra_Library::BHRZ03_-
Certificate, 342
num_rows
Parma_Polyhedra_Library::Bit_Matrix, 349
Parma_Polyhedra_Library::DB_Matrix, 614
Parma_Polyhedra_Library::Grid_Generator_-
System, 869
Parma_Polyhedra_Library::Matrix, 1090
Parma_Polyhedra_Library::OR_Matrix, 1237
numer_denom
Parma_Polyhedra_Library, 151

objective_function
Parma_Polyhedra_Library::MIP_Problem,
1119
obtain_sorted_constraints
Parma_Polyhedra_Library::Polyhedron, 1482
obtain_sorted_constraints_with_sat_c
Parma_Polyhedra_Library::Polyhedron, 1483
obtain_sorted_generators
Parma_Polyhedra_Library::Polyhedron, 1483
obtain_sorted_generators_with_sat_g
Parma_Polyhedra_Library::Polyhedron, 1483
Octagonal_Shape
Parma_Polyhedra_Library::Octagonal_Shape,
1160–1164
Octagonal_Shape.cc, 1858
Octagonal_Shape.defs.hh, 1858
PPL_IN_Octagonal_Shape_CLASS, 1860
Octagonal_Shape.inlines.hh, 1861
Octagonal_Shape.templates.hh, 1862
Octagonal_Shape.types.hh, 1863
OK
Parma_Polyhedra_Library::BD_Shape, 313
Parma_Polyhedra_Library::BD_-
Shape::Status, 1618
Parma_Polyhedra_Library::BHRZ03_-
Certificate, 341
Parma_Polyhedra_Library::Bit_Matrix, 350
Parma_Polyhedra_Library::Bit_Row, 359
Parma_Polyhedra_Library::Box, 411
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Checked_Number,
461
Parma_Polyhedra_Library::Congruence, 498

Parma_Polyhedra_Library::Congruence_-
System, 518
Parma_Polyhedra_Library::Constraint, 578
Parma_Polyhedra_Library::Constraint_-
System, 601
Parma_Polyhedra_Library::DB_Matrix, 614
Parma_Polyhedra_Library::DB_Row, 629
Parma_Polyhedra_Library::Determinate, 641
Parma_Polyhedra_Library::Generator, 716
Parma_Polyhedra_Library::Generator_-
System, 740
Parma_Polyhedra_Library::Grid, 809
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_Library::Grid_Certificate,
834
Parma_Polyhedra_Library::Grid_Generator,
850
Parma_Polyhedra_Library::Grid_Generator_-
System, 870
Parma_Polyhedra_Library::Interval, 921
Parma_Polyhedra_Library::Linear_-
Expression, 1030
Parma_Polyhedra_Library::Linear_Row, 1051
Parma_Polyhedra_Library::Linear_System,
1069
Parma_Polyhedra_Library::Matrix, 1092
Parma_Polyhedra_Library::MIP_Problem,
1119
Parma_Polyhedra_Library::Octagonal_Shape,
1201
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1646
Parma_Polyhedra_Library::OR_Matrix, 1237
Parma_Polyhedra_Library::Partially_-
Reduced_Product, 1278
Parma_Polyhedra_Library::PIP_Decision_-
Node, 1294
Parma_Polyhedra_Library::PIP_Problem,
1315
Parma_Polyhedra_Library::PIP_Solution_-
Node, 1327
Parma_Polyhedra_Library::PIP_Solution_-
Node::Tableau, 1657
Parma_Polyhedra_Library::PIP_Tree_Node,
1342
Parma_Polyhedra_Library::PIP_Tree_-
Node::Artificial_Parameter, 257
Parma_Polyhedra_Library::Pointset_-
Powerset, 1387
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1404
Parma_Polyhedra_Library::Poly_Gen_-
Relation, 1411
Parma_Polyhedra_Library::Polyhedron, 1484
Parma_Polyhedra_-
Library::Polyhedron::Status, 1635
Parma_Polyhedra_Library::Powerset, 1528
Parma_Polyhedra_Library::Row, 1564
Parma_Polyhedra_Library::Variable, 1668
Parma_Polyhedra_Library::Variables_Set,
1674
Status, 1625
old_rounding_direction
Parma_Polyhedra_Library::Init, 900
omega_reduce
Parma_Polyhedra_Library::Powerset, 1528
one_affine_ranking_function_MS
Parma_Polyhedra_Library, 151
Parma_Polyhedra_-
Library::Implementation::Termination,
230
one_affine_ranking_function_MS_2
Parma_Polyhedra_Library, 152
one_affine_ranking_function_PR
Parma_Polyhedra_Library, 153
Parma_Polyhedra_-
Library::Implementation::Termination,
230
one_affine_ranking_function_PR_2
Parma_Polyhedra_Library, 153
one_affine_ranking_function_PR_original
Parma_Polyhedra_-
Library::Implementation::Termination,
231
op_assign_
Parma_Polyhedra_-
Library::Determinate::Binary_Operator_-
Assign_Lifter, 343
open
Parma_Polyhedra_Library::Interval_Info_-
Null_Open, 943
operator delete
Parma_Polyhedra_Library::DB_Row_Impl_-
Handler::Impl, 895
Parma_Polyhedra_Library::Row_Impl_-
Handler::Impl, 887
operator new
Parma_Polyhedra_Library::DB_Row_Impl_-
Handler::Impl, 895
Parma_Polyhedra_Library::Row_Impl_-
Handler::Impl, 887
operator T
Parma_Polyhedra_Library::Checked_Number,
461
operator<
Parma_Polyhedra_Library::Checked_Number,
472
Parma_Polyhedra_Library::Constraint, 582

Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 248
operator<<
Parma_Polyhedra_Library, 156
Parma_Polyhedra_Library::BD_Shape, 333
Parma_Polyhedra_Library::Box, 433
Parma_Polyhedra_Library::Checked_Number, 472
Parma_Polyhedra_Library::Congruence, 502
Parma_Polyhedra_Library::Congruence_-System, 522
Parma_Polyhedra_Library::Constraint, 583
Parma_Polyhedra_Library::Constraint_-System, 605
Parma_Polyhedra_Library::DB_Matrix, 617
Parma_Polyhedra_Library::Determinate, 643
Parma_Polyhedra_Library::Generator, 722
Parma_Polyhedra_Library::Generator_-System, 744
Parma_Polyhedra_Library::Grid, 829
Parma_Polyhedra_Library::Grid_Generator, 855, 856
Parma_Polyhedra_Library::Grid_Generator_-System, 875
Parma_Polyhedra_Library::IO_Operators, 240
Parma_Polyhedra_Library::Linear_-Expression, 1036
Parma_Polyhedra_Library::MIP_Problem, 1127
Parma_Polyhedra_Library::Octagonal_Shape, 1222
Parma_Polyhedra_Library::OR_Matrix, 1243
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1286
Parma_Polyhedra_Library::PIP_Problem, 1318
Parma_Polyhedra_Library::PIP_Tree_Node, 1345
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 258
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1412
Parma_Polyhedra_Library::Polyhedron, 1510
Parma_Polyhedra_Library::Powerset, 1532
Parma_Polyhedra_Library::Variable, 1670
Parma_Polyhedra_Library::Variables_Set, 1676
operator<=
Parma_Polyhedra_Library::Checked_Number, 472
Parma_Polyhedra_Library::Constraint, 583
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 248
operator>
Parma_Polyhedra_Library::Checked_Number, 472
Parma_Polyhedra_Library::Constraint, 584, 585
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 249
operator>>
Parma_Polyhedra_Library::Checked_Number, 473
operator>=
Parma_Polyhedra_Library::Checked_Number, 472
Parma_Polyhedra_Library::Constraint, 585, 586
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 249
operator*
Parma_Polyhedra_Library, 153, 154
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 541
Parma_Polyhedra_Library::const_iterator_-to_const, 558
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 537
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 532
Parma_Polyhedra_Library::Generator_-System::const_iterator, 547
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 552
Parma_Polyhedra_Library::iterator_to_const, 994
Parma_Polyhedra_Library::Linear_-Expression, 1033
Parma_Polyhedra_Library::Matrix::const_-iterator, 527
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 246
operator*=
Parma_Polyhedra_Library::Checked_Number, 462
Parma_Polyhedra_Library::Interval, 921
Parma_Polyhedra_Library::Linear_-Expression, 1033
operator()
Parma_Polyhedra_Library::BHRZ03_-Certificate::Compare, 481
Parma_Polyhedra_Library::Bit_Matrix::Bit_-Row_Less_Than, 364
Parma_Polyhedra_-Library::Determinate::Binary_Operator_-

Assign_Lifter, 343
Parma_Polyhedra_Library::Grid_-Certificate::Compare, 483
Parma_Polyhedra_Library::H79_-Certificate::Compare, 482
Parma_Polyhedra_Library::Limited_-Widening_Function, 1016
Parma_Polyhedra_Library::Linear_-System::Row_Less_Than, 1571
Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Sign, 1662
Parma_Polyhedra_-Library::Variable::Compare, 483
Parma_Polyhedra_Library::Widening_-Function, 1679
operator+
Parma_Polyhedra_Library, 154
Parma_Polyhedra_Library::Checked_Number, 471
Parma_Polyhedra_Library::Linear_-Expression, 1033, 1034
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 246
operator++
Parma_Polyhedra_Library::Checked_Number, 462
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 542
Parma_Polyhedra_Library::const_iterator_-to_const, 558
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 537
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 532, 533
Parma_Polyhedra_Library::Generator_-System::const_iterator, 547
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 552, 553
Parma_Polyhedra_Library::iterator_to_const, 995
Parma_Polyhedra_Library::Matrix::const_-iterator, 528
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 246
operator+=
Parma_Polyhedra_Library::Checked_Number, 462, 463
Parma_Polyhedra_Library::Interval, 921
Parma_Polyhedra_Library::Linear_-Expression, 1034, 1035
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 247
operator-
Parma_Polyhedra_Library, 155
Parma_Polyhedra_Library::Checked_Number, 471
Parma_Polyhedra_Library::Linear_-Expression, 1035, 1036
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 247
Parma_Polyhedra_Library::Poly_Con_-Relation, 1405
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1412
operator->
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 542
Parma_Polyhedra_Library::const_iterator_-to_const, 559
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 537
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 533
Parma_Polyhedra_Library::Generator_-System::const_iterator, 547
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 553
Parma_Polyhedra_Library::iterator_to_const, 995
Parma_Polyhedra_Library::Matrix::const_-iterator, 528
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 248
operator--
Parma_Polyhedra_Library::Checked_Number, 463
Parma_Polyhedra_Library::const_iterator_-to_const, 558, 559
Parma_Polyhedra_Library::iterator_to_const, 995
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 247, 248
operator-=
Parma_Polyhedra_Library::Checked_Number, 463
Parma_Polyhedra_Library::Interval, 922
Parma_Polyhedra_Library::Linear_-Expression, 1036
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 248
operator/
Parma_Polyhedra_Library, 155, 156
Parma_Polyhedra_Library::Congruence, 501, 502
operator/=
Parma_Polyhedra_Library::Checked_Number, 464
Parma_Polyhedra_Library::Congruence, 498

Parma_Polyhedra_Library::Interval, 922
operator=
Parma_Polyhedra_Library::BD_Shape, 314
Parma_Polyhedra_Library::Bit_Matrix, 350
Parma_Polyhedra_Library::Bit_Row, 359
Parma_Polyhedra_Library::Box, 412
Parma_Polyhedra_Library::C_Polyhedron, 443
Parma_Polyhedra_Library::Checked_Number, 464
Parma_Polyhedra_Library::Congruence, 498
Parma_Polyhedra_Library::Congruence_System, 518
Parma_Polyhedra_Library::Congruence_System::const_iterator, 542
Parma_Polyhedra_Library::Constraint, 578
Parma_Polyhedra_Library::Constraint_System, 602
Parma_Polyhedra_Library::Constraint_System::const_iterator, 537
Parma_Polyhedra_Library::DB_Matrix, 615
Parma_Polyhedra_Library::DB_Matrix::const_iterator, 533
Parma_Polyhedra_Library::DB_Row, 629
Parma_Polyhedra_Library::DB_Row_Impl_Handler, 634
Parma_Polyhedra_Library::DB_Row_Impl_Handler::Impl, 895
Parma_Polyhedra_Library::Determinate, 641
Parma_Polyhedra_Library::Determinate::Rep, 1545
Parma_Polyhedra_Library::Generator, 716
Parma_Polyhedra_Library::Generator_System, 740
Parma_Polyhedra_Library::Generator_System::const_iterator, 548
Parma_Polyhedra_Library::Grid, 809
Parma_Polyhedra_Library::Grid_Generator, 850
Parma_Polyhedra_Library::Grid_Generator_System, 870
Parma_Polyhedra_Library::Grid_Generator_System::const_iterator, 553
Parma_Polyhedra_Library::Interval, 922
Parma_Polyhedra_Library::Linear_System, 1070
Parma_Polyhedra_Library::Matrix, 1092
Parma_Polyhedra_Library::Matrix::const_iterator, 528
Parma_Polyhedra_Library::MIP_Problem, 1119
Parma_Polyhedra_Library::NNC_Polyhedron, 1142
Parma_Polyhedra_Library::Octagonal_Shape, 1201
Parma_Polyhedra_Library::OR_Matrix, 1238
Parma_Polyhedra_Library::OR_Matrix::any_row_iterator, 249
Parma_Polyhedra_Library::OR_Matrix::Pseudo_Row, 1538
Parma_Polyhedra_Library::Partially_Reduced_Product, 1278
Parma_Polyhedra_Library::PIP_Problem, 1315
Parma_Polyhedra_Library::Pointset_Powerset, 1387, 1388
Parma_Polyhedra_Library::Polyhedron, 1485
Parma_Polyhedra_Library::Powerset, 1529
Parma_Polyhedra_Library::Row, 1564
Parma_Polyhedra_Library::Row_Impl_Handler, 1570
Parma_Polyhedra_Library::Row_Impl_Handler::Impl, 887
operator==
Parma_Polyhedra_Library, 156
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Bit_Matrix, 353
Parma_Polyhedra_Library::Bit_Row, 362
Parma_Polyhedra_Library::Box, 433
Parma_Polyhedra_Library::Checked_Number, 472
Parma_Polyhedra_Library::Congruence, 502
Parma_Polyhedra_Library::Congruence_System, 522
Parma_Polyhedra_Library::Congruence_System::const_iterator, 542
Parma_Polyhedra_Library::const_iterator_to_const, 559, 560
Parma_Polyhedra_Library::Constraint, 584
Parma_Polyhedra_Library::Constraint_System, 605
Parma_Polyhedra_Library::Constraint_System::const_iterator, 538
Parma_Polyhedra_Library::DB_Matrix, 618
Parma_Polyhedra_Library::DB_Matrix::const_iterator, 533
Parma_Polyhedra_Library::DB_Row, 632
Parma_Polyhedra_Library::Determinate, 643
Parma_Polyhedra_Library::Generator, 722
Parma_Polyhedra_Library::Generator_System, 744
Parma_Polyhedra_Library::Generator_System::const_iterator, 548
Parma_Polyhedra_Library::Grid, 829
Parma_Polyhedra_Library::Grid_Generator, 856

Parma_Polyhedra_Library::Grid_Generator_-System, 875
Parma_Polyhedra_Library::Grid_Generator_-System::const_iterator, 553
Parma_Polyhedra_Library::iterator_to_const, 996
Parma_Polyhedra_Library::Linear_Row, 1055
Parma_Polyhedra_Library::Linear_-Row::Flags, 666
Parma_Polyhedra_Library::Linear_System, 1077
Parma_Polyhedra_Library::Matrix, 1095
Parma_Polyhedra_Library::Matrix::const_-iterator, 528
Parma_Polyhedra_Library::Octagonal_Shape, 1222
Parma_Polyhedra_Library::OR_Matrix, 1243
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 249
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1286
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 257
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1412
Parma_Polyhedra_Library::Polyhedron, 1510
Parma_Polyhedra_Library::Powerset, 1532
Parma_Polyhedra_Library::Row, 1568
Parma_Polyhedra_Library::Row::Flags, 659
operator%=
Parma_Polyhedra_Library::Checked_Number, 461, 462
Parma_Polyhedra_Library::Congruence, 501
operator&&
Parma_Polyhedra_Library::Poly_Con_-Relation, 1405
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1412
opt_mode
Parma_Polyhedra_Library::MIP_Problem, 1130
optimal_value
Parma_Polyhedra_Library::MIP_Problem, 1120
Optimization_Mode
PPL_CXX_interface, 103
optimization_mode
Parma_Polyhedra_Library::MIP_Problem, 1120
OPTIMIZED
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::PIP_Problem, 1309
OPTIMIZED_MIP_PROBLEM
PPL_CXX_interface, 103
OPTIMIZED_PIP_PROBLEM
PPL_CXX_interface, 103
optimizing_point
Parma_Polyhedra_Library::MIP_Problem, 1120
optimizing_solution
Parma_Polyhedra_Library::PIP_Problem, 1315
OR_Matrix
Parma_Polyhedra_Library::OR_Matrix, 1233, 1234, 1243
Parma_Polyhedra_Library::OR_-Matrix::Pseudo_Row, 1538
OR_Matrix.defs.hh, 1863
PPL_OR_MATRIX_EXTRA_DEBUG, 1865
OR_Matrix.inlines.hh, 1865
OR_Matrix.templates.hh, 1866
OR_Matrix.types.hh, 1868
output
Parma_Polyhedra_Library::Checked_Number, 473
output_char
Parma_Polyhedra_Library::Checked, 202
output_float
Parma_Polyhedra_Library::Checked, 202
output_function_type
Parma_Polyhedra_Library::Variable, 1666
output_int
Parma_Polyhedra_Library::Checked, 202
output_mpq
Parma_Polyhedra_Library::Checked, 202
output_mpz
Parma_Polyhedra_Library::Checked, 202
output_restriction
Parma_Polyhedra_Library, 156, 157
overflow
Parma_Polyhedra_Library::stdiobuf, 1651
OVERFLOW_IMPOSSIBLE
PPL_CXX_interface, 101
OVERFLOW_UNDEFINED
PPL_CXX_interface, 101
OVERFLOW_WRAPS
PPL_CXX_interface, 101
owner_
Parma_Polyhedra_Library::PIP_Tree_Node, 1346

pairwise_apply_assign
Parma_Polyhedra_Library::Powerset, 1529
pairwise_reduce

Parma_Polyhedra_Library::Pointset_-
Powerset, 1388
PARAMETER
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Grid_Generator, 843
parameter
Parma_Polyhedra_Library::Grid_Generator, 851
parameter_space_dimensions
Parma_Polyhedra_Library::PIP_Problem, 1315
parameters
Parma_Polyhedra_Library::PIP_Problem, 1320
parametric_values
Parma_Polyhedra_Library::PIP_Solution_-
Node, 1327
parent
Parma_Polyhedra_Library::PIP_Tree_Node, 1342
parent_
Parma_Polyhedra_Library::PIP_Tree_Node, 1346
parent_merge
Parma_Polyhedra_Library::PIP_Tree_Node, 1342
Parma_Polyhedra_Library
T_MAYBE, 122
T_NO, 122
T_YES, 122
Parma_Polyhedra_Library::Bool
value, 365
Parma_Polyhedra_Library::Constraint
EQUALITY, 571
NONSTRICT_INEQUALITY, 571
STRICT_INEQUALITY, 571
Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type >
value, 655
Parma_Polyhedra_Library::Generator
CLOSURE_POINT, 709
LINE, 709
POINT, 709
RAY, 709
Parma_Polyhedra_Library::Grid
CON_VIRTUAL, 766
EQUALITY, 766
GEN_VIRTUAL, 766
LINE, 766
PARAMETER, 766
PROPER_CONGRUENCE, 766
TVB_DONT_KNOW, 766
TVB_FALSE, 766
TVB_TRUE, 766
Parma_Polyhedra_Library::Grid_Generator
LINE, 843
PARAMETER, 843
POINT, 844
Parma_Polyhedra_Library::Interval_NS::Property
CARDINALITY_0_, 1535
CARDINALITY_1_, 1535
CARDINALITY_IS_, 1535
Parma_Polyhedra_Library::Is_Same_Or_Derived
value, 982
Parma_Polyhedra_Library::Linear_Row
LINE_OR_EQUALITY, 1045
RAY_OR_POINT_OR_INEQUALITY, 1045
Parma_Polyhedra_Library::MIP_Problem
OPTIMIZED, 1106
PARTIALLY_SATISFIABLE, 1106
PRICING, 1106
PRICING_STEEPEST_EDGE_EXACT, 1106
PRICING_STEEPEST_EDGE_FLOAT, 1106
PRICING_TEXTBOOK, 1106
SATISFIABLE, 1106
UNBOUNDED, 1106
UNSATISFIABLE, 1106
Parma_Polyhedra_Library::PIP_Problem
CONTROL_PARAMETER_NAME_SIZE, 1308
CONTROL_PARAMETER_VALUE_SIZE, 1308
CUTTING_STRATEGY, 1308
CUTTING_STRATEGY_ALL, 1308
CUTTING_STRATEGY_DEEPEST, 1308
CUTTING_STRATEGY_FIRST, 1308
OPTIMIZED, 1309
PARTIALLY_SATISFIABLE, 1309
PIVOT_ROW_STRATEGY, 1308
PIVOT_ROW_STRATEGY_FIRST, 1308
PIVOT_ROW_STRATEGY_MAX_-
COLUMN, 1308
UNSATISFIABLE, 1309
Parma_Polyhedra_Library::PIP_Solution_Node
MIXED, 1324
NEGATIVE, 1324
POSITIVE, 1324
UNKNOWN, 1324
ZERO, 1324
Parma_Polyhedra_Library::Polyhedron
TVB_DONT_KNOW, 1432
TVB_FALSE, 1432
TVB_TRUE, 1432
Parma_Polyhedra_Library, 104
abs_assign, 122
add_mul_assign, 123

add_restriction, 123
addmod, 124
all_affine_quasi_ranking_functions_MS, 124
all_affine_quasi_ranking_functions_MS_2, 125
all_affine_ranking_functions_MS, 125
all_affine_ranking_functions_MS_2, 126
all_affine_ranking_functions_PR, 127
all_affine_ranking_functions_PR_2, 127
ascii_dump, 127
ascii_load, 128
assign_r, 128
assign_rem, 129
assign_restriction, 129, 130
banner, 130
check_empty_arg, 130
check_result, 130
cmp, 131
Coefficient_constants_finalize, 131
Coefficient_constants_initialize, 131
Coefficient_one, 131
Coefficient_zero, 131
combine, 131
compute_capacity, 132
construct, 132
contains_restriction, 133
diff_restriction, 133, 134
div_2exp_assign, 134
div_restriction, 134
div_round_up, 134
eq_restriction, 135
equal, 135
exact_div_assign, 135
external_memory_in_bytes, 136
extract_interval_congruence, 137
f_is_empty, 137
f_is_singleton, 137
f_OK, 137
finalize, 138
gcd_assign, 138
gcdext_assign, 138
GMP_Integer, 122
greater_or_equal, 138
greater_than, 139
initialize, 139
intersect_restriction, 139
interval_relation, 139
irrational_precision, 140
is_additive_inverse, 140
is_canonical, 141
is_even, 141
is_infinity, 141
is_integer, 141
is_minus_infinity, 142
is_not_a_number, 142, 143
is_plus_infinity, 143
is_singleton_integer, 144
isqrt, 144
join_restriction, 144, 145
lcm_assign, 145
less_or_equal, 145
less_than, 145
low_bits_mask, 145
max_assign, 146
maybe_abandon, 146
maybe_assign, 146
maybe_check_fpu_inexact, 146
maybe_reset_fpu_inexact, 147
min_assign, 147
MINUS_INFINITY, 165
minus_infinity, 147
mul_2exp_assign, 147
mul_restriction, 147, 148
mulmod, 148
neg_assign, 148
neg_restriction, 149, 150
normalize2, 150
not_a_dimension, 150
NOT_A_NUMBER, 165
not_a_number, 150
not_equal, 151
numer_denom, 151
one_affine_ranking_function_MS, 151
one_affine_ranking_function_MS_2, 152
one_affine_ranking_function_PR, 153
one_affine_ranking_function_PR_2, 153
operator<<, 156
operator∗, 153, 154
operator+, 154
operator-, 155
operator/, 155, 156
operator==, 156
output_restriction, 156, 157
PIP_Tree, 122
PLUS_INFINITY, 165
plus_infinity, 157
raw_value, 157
rem_assign, 158
restore_pre_PPL_rounding, 158
rounding_dir, 158
set_integer, 158
set_irrational_precision, 158
set_rounding_for_PPL, 159
set_unrestricted, 159
shrink_to_congruence_no_check, 159
sqrt_assign, 159
sub_mul_assign, 160
sub_restriction, 160

submod, 160
swap, 161
termination_test_MS, 161
termination_test_MS_2, 162
termination_test_PR, 163
termination_test_PR_2, 163
Ternary, 122
throw_result_exception, 163
total_memory_in_bytes, 163, 164
version, 164
version_beta, 164
version_major, 164
version_minor, 164
version_revision, 165
Parma_Polyhedra_Library::BD_Shape, 258
~BD_Shape, 276
add_congruence, 277
add_congruences, 277
add_constraint, 277
add_constraints, 278
add_dbm_constraint, 278, 279
add_recycled_congruences, 279
add_recycled_constraints, 279
add_space_dimensions_and_embed, 280
add_space_dimensions_and_project, 280
affine_dimension, 281
affine_image, 281
affine_preimage, 282
ascii_dump, 282, 283
ascii_load, 283
BD_Shape, 272–276
BFT00_upper_bound_assign_if_exact, 283
BHMZ05_widening_assign, 284
BHZ09_upper_bound_assign_if_exact, 284
bounded_affine_image, 285
bounded_affine_preimage, 285
bounds, 286
bounds_from_above, 287
bounds_from_below, 287
can_recycle_congruence_systems, 287
can_recycle_constraint_systems, 287
CC76_extrapolation_assign, 287, 288
CC76_narrowing_assign, 288
coefficient_type, 272
coefficient_type_base, 272
compute_leader_indices, 331
compute_leaders, 289
compute_predecessors, 289
concatenate_assign, 290
congruences, 290
constrains, 290
constraints, 291
contains, 291
contains_integer_point, 292
dbm, 336
deduce_u_minus_v_bounds, 292
deduce_v_minus_u_bounds, 292
difference_assign, 293
drop_some_non_integer_points, 293, 294
drop_some_non_integer_points_helper, 294
euclidean_distance_assign, 331
expand_space_dimension, 294
external_memory_in_bytes, 295
extract_bounded_difference, 332
fold_space_dimensions, 295
forget_all_dbm_constraints, 296
forget_binary_dbm_constraints, 296
frequency, 296
generalized_affine_image, 297, 298
generalized_affine_preimage, 299
get_limiting_shape, 300
H79_widening_assign, 300
hash_code, 301
incremental_shortest_path_closure_assign, 301
integer_upper_bound_assign_if_exact, 302
intersection_assign, 302
is_bounded, 302
is_discrete, 303
is_disjoint_from, 303
is_empty, 303
is_shortest_path_reduced, 304
is_topologically_closed, 304
is_universe, 304
l_infinity_distance_assign, 332, 333
limited_BHMZ05_extrapolation_assign, 304
limited_CC76_extrapolation_assign, 305
limited_H79_extrapolation_assign, 305
map_space_dimensions, 306
marked_empty, 307
marked_shortest_path_closed, 308
marked_shortest_path_reduced, 308
marked_zero_dim_univ, 309
max_min, 309
max_space_dimension, 310
maximize, 310, 311
minimize, 311, 312
minimized_congruences, 312
minimized_constraints, 313
N, 272
OK, 313
operator<<, 333
operator=, 314
operator==, 334
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Box, 334
Parma_Polyhedra_Library::euclidean_-distance_assign, 334

Parma_Polyhedra_Library::IO_-
Operators::operator<<, 334
Parma_Polyhedra_Library::l_infinity_-
distance_assign, 334
Parma_Polyhedra_Library::rectilinear_-
distance_assign, 334
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Polyhedron, 1510
print, 314
rectilinear_distance_assign, 335
redundancy_dbm, 337
refine, 314
refine_no_check, 315
refine_with_congruence, 316
refine_with_congruences, 316
refine_with_constraint, 317
refine_with_constraints, 317
relation_with, 317, 318
remove_higher_space_dimensions, 319
remove_space_dimensions, 319
reset_shortest_path_closed, 320
reset_shortest_path_reduced, 320
set_empty, 321
set_shortest_path_closed, 321
set_shortest_path_reduced, 321
set_zero_dim_univ, 322
shortest_path_closure_assign, 322
shortest_path_reduction_assign, 323
simplify_using_context_assign, 323
space_dimension, 324
status, 337
strictly_contains, 325
swap, 325, 335
throw_dimension_incompatible, 325, 326
throw_expression_too_complex, 327
throw_generic, 327
time_elapse_assign, 327
topological_closure_assign, 328
total_memory_in_bytes, 328
unconstrain, 328, 329
upper_bound_assign, 329
upper_bound_assign_if_exact, 329
widening_assign, 330
wrap_assign, 330
Parma_Polyhedra_Library::BD_Shape::Status,
1615
ascii_dump, 1618
ascii_load, 1618
EMPTY, 1621
flags, 1621
flags_t, 1618
OK, 1618
print, 1619
reset, 1619
reset_empty, 1619
reset_shortest_path_closed, 1619
reset_shortest_path_reduced, 1619
reset_zero_dim_univ, 1619
set, 1619
set_empty, 1619
set_shortest_path_closed, 1620
set_shortest_path_reduced, 1620
set_zero_dim_univ, 1620
SHORTEST_PATH_CLOSED, 1621
SHORTEST_PATH_REDUCED, 1621
Status, 1618
test_all, 1620
test_any, 1620
test_empty, 1620
test_shortest_path_closed, 1620
test_shortest_path_reduced, 1621
test_zero_dim_univ, 1621
ZERO_DIM_UNIV, 1621
Parma_Polyhedra_Library::BHRZ03_Certificate,
337
∼BHRZ03_Certificate, 340
affine_dim, 341
BHRZ03_Certificate, 339
compare, 340
is_stabilizing, 340
lin_space_dim, 341
num_constraints, 341
num_points, 342
num_rays_null_coord, 342
OK, 341
Parma_Polyhedra_Library::Polyhedron, 1510
Parma_Polyhedra_Library::BHRZ03_-
Certificate::Compare, 481
operator(), 481
Parma_Polyhedra_Library::Bit_Matrix, 344
∼Bit_Matrix, 347
add_recycled_row, 347
ascii_dump, 347, 348
ascii_load, 348
Bit_Matrix, 347
check_sorted, 348
clear, 348
columns_erase_to_end, 348
external_memory_in_bytes, 349
max_num_rows, 349
num_columns, 349
num_rows, 349
OK, 350
operator=, 350
operator==, 353
Parma_Polyhedra_Library::Linear_-
System::sort_and_remove_with_sat,
353

print, 350
resize, 351
row_size, 353
rows, 354
rows_erase_to_end, 351
sort_rows, 351
sorted_contains, 351
swap, 352, 353
total_memory_in_bytes, 352
transpose, 352
transpose_assign, 352
Parma_Polyhedra_Library::Bit_Matrix::Bit_Row_-Less_Than, 363
operator(), 364
Parma_Polyhedra_Library::Bit_Row, 354
∼Bit_Row, 357
Bit_Row, 357
clear, 357
clear_from, 358
compare, 361
count_ones, 358
empty, 358
external_memory_in_bytes, 358
first, 359
iter_swap, 361
last, 359
next, 359
OK, 359
operator=, 359
operator==, 362
prev, 360
set, 360
set_difference, 362
set_intersection, 362
set_union, 362
set_until, 360
strict_subset, 362
subset_or_equal, 363
swap, 360, 363
total_memory_in_bytes, 361
union_helper, 361
vec, 363
Parma_Polyhedra_Library::Bool, 364
anonymous_enum, 365
Parma_Polyhedra_Library::Boundary_Value, 365
type, 365
Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_-Derived< Interval_Base, T >::value >::type >, 366
type, 366
Parma_Polyhedra_Library::Box, 366
add_congruence, 383
add_congruence_no_check, 384
add_congruences, 384
add_congruences_no_check, 384
add_constraint, 385
add_constraint_no_check, 385
add_constraints, 385
add_constraints_no_check, 386
add_interval_constraint_no_check, 386
add_recycled_congruences, 386
add_recycled_constraints, 387
add_space_dimensions_and_embed, 387
add_space_dimensions_and_project, 388
affine_dimension, 388
affine_image, 388
affine_preimage, 389
ascii_dump, 390
ascii_load, 390
bounded_affine_image, 390
bounded_affine_preimage, 391
bounds, 391
bounds_from_above, 392
bounds_from_below, 392
Box, 378–383
can_recycle_congruence_systems, 392
can_recycle_constraint_systems, 392
CC76_narrowing_assign, 393
CC76_widening_assign, 393, 394
check_empty, 394
concatenate_assign, 394
congruences, 395
constrains, 395
constraints, 395
contains, 396
contains_integer_point, 396
difference_assign, 396
drop_some_non_integer_points, 397
euclidean_distance_assign, 431
expand_space_dimension, 398
external_memory_in_bytes, 398
extract_interval_constraint, 431
fold_space_dimensions, 398
frequency, 399
generalized_affine_image, 399, 400
generalized_affine_preimage, 401
get_interval, 402
get_limiting_box, 402
get_lower_bound, 402
get_upper_bound, 403
intersection_assign, 403
interval_type, 378
is_bounded, 404
is_discrete, 404
is_disjoint_from, 404
is_empty, 404
is_topologically_closed, 405

is_universe, 405
l_infinity_distance_assign, 432
l_m_distance_assign, 433
limited_CC76_extrapolation_assign, 405
map_space_dimensions, 406
marked_empty, 407
max_min, 407, 408
max_space_dimension, 409
maximize, 409
minimize, 410
minimized_congruences, 411
minimized_constraints, 411
OK, 411
operator<<, 433
operator=, 412
operator==, 433
Parma_Polyhedra_Library::Box, 433
Parma_Polyhedra_Library::IO_-
Operators::operator<<, 434
Parma_Polyhedra_Library::l_m_distance_-
assign, 434
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Box, 433
Parma_Polyhedra_Library::Grid, 829
Parma_Polyhedra_Library::Octagonal_Shape,
1223
Parma_Polyhedra_Library::Polyhedron, 1510
print, 412
propagate_constraint, 412
propagate_constraint_no_check, 413
propagate_constraints, 414
propagate_constraints_no_check, 415
rectilinear_distance_assign, 434
refine_interval_no_check, 415
refine_no_check, 416, 417
refine_with_congruence, 417
refine_with_congruences, 418
refine_with_constraint, 418
refine_with_constraints, 419
relation_with, 419, 420
remove_higher_space_dimensions, 421
remove_space_dimensions, 421
reset_empty_up_to_date, 421
seq, 435
Sequence, 378
set_empty, 422
set_empty_up_to_date, 422
set_interval, 422
set_nonempty, 423
simplify_using_context_assign, 423
space_dimension, 423
status, 435
strictly_contains, 424
swap, 424
throw_constraint_incompatible, 425
throw_dimension_incompatible, 425, 426
throw_expression_too_complex, 427
throw_generic, 427
throw_space_dimension_overflow, 427
time_elapse_assign, 427
Tmp_Interval_Type, 378
topological_closure_assign, 428
total_memory_in_bytes, 428
unconstrain, 428
upper_bound_assign, 429
upper_bound_assign_if_exact, 429
widening_assign, 429
wrap_assign, 430
Parma_Polyhedra_Library::Box::Status, 1610
ascii_dump, 1612
ascii_load, 1612
EMPTY, 1615
EMPTY_UP_TO_DATE, 1615
flags, 1615
flags_t, 1611
NONE, 1615
OK, 1613
print, 1613
reset, 1613
reset_empty, 1613
reset_empty_up_to_date, 1613
reset_universe, 1613
set, 1613
set_empty, 1613
set_empty_up_to_date, 1614
set_universe, 1614
Status, 1612
test_all, 1614
test_any, 1614
test_empty, 1614
test_empty_up_to_date, 1614
test_universe, 1614
UNIVERSE, 1615
Parma_Polyhedra_Library::C_Polyhedron, 436
~C_Polyhedron, 442
C_Polyhedron, 438–442
operator=, 443
poly_hull_assign_if_exact, 443
upper_bound_assign_if_exact, 443
Parma_Polyhedra_Library::Check_Overflow_-
Policy, 444
const_bool_nodef, 444–446
Parma_Polyhedra_Library::Checked, 166
abs_float, 178
abs_generic, 178
abs_mpq, 178
abs_mpz, 178
add_2exp_float, 179

add_2exp_mpq, 179
add_2exp_mpz, 179
add_2exp_signed_int, 179
add_2exp_unsigned_int, 179
add_float, 179
add_int_larger, 180
add_mpq, 180
add_mpz, 180
add_mul_float, 180
add_mul_int, 180
add_mul_mpq, 180
add_mul_mpz, 181
add_signed_int, 181
add_unsigned_int, 181
assign_exact, 181
assign_float_float, 181
assign_float_float_exact, 181
assign_float_float_inexact, 182
assign_float_int, 182
assign_float_int_inexact, 182
assign_float_mpq, 182
assign_float_mpz, 182
assign_int_float, 183
assign_int_mpq, 183
assign_mpq_float, 183
assign_mpq_long_double, 183
assign_mpq_signed_int, 183
assign_mpq_unsigned_int, 183
assign_mpz_float, 184
assign_mpz_long_double, 184
assign_mpz_mpq, 184
assign_mpz_signed_int, 184
assign_mpz_unsigned_int, 184
assign_nan, 185
assign_signed_int_mpz, 185
assign_signed_int_signed_int, 185
assign_signed_int_unsigned_int, 185
assign_special_float, 185
assign_special_int, 185
assign_special_mpq, 186
assign_special_mpz, 186
assign_unsigned_int_mpz, 186
assign_unsigned_int_signed_int, 186
assign_unsigned_int_unsigned_int, 186
bool, 220
ceil_float, 186
ceil_mpq, 187
classify_float, 187
classify_int, 187
classify_mpq, 187
classify_mpz, 187
cmp_float, 187
cmp_generic, 188
cmp_mp, 188
const, 220
construct_mpq_base, 188
construct_mpq_float, 188
construct_mpz_base, 188
construct_mpz_float, 188
copy_generic, 189
copy_mpz, 189
div_2exp_float, 189
div_2exp_mpq, 189
div_2exp_mpz, 189
div_2exp_signed_int, 189
div_2exp_unsigned_int, 190
div_float, 190
div_mpq, 190
div_mpz, 190
div_signed_int, 190
div_unsigned_int, 191
eq, 191, 220
eq_p, 191
floor_float, 191
floor_mpq, 191
fma, 191, 192
fpu_direct_rounding, 192
fpu_inverse_rounding, 192
From, 220
From1, 221
From2, 221
gcd_exact, 192
gcd_exact_noabs, 192
gcd_mpz, 193
gcdext_exact, 193
gcdext_mpz, 193
get_digit, 193
get_mp_size, 193
idiv_float, 193
idiv_mpq, 194
idiv_mpz, 194
idiv_signed_int, 194
idiv_unsigned_int, 194
input_generic, 194
input_mpq, 195
int, 221
int_fast16_t, 177
int_fast32_t, 177
int_fast64_t, 177
irrational_precision, 221
is_inf_float, 195
is_int_float, 195
is_int_int, 195
is_int_mpq, 195
is_int_mpz, 196
is_minf_float, 196
is_minf_int, 196
is_minf_mpq, 196

is_minf_mpz, 196
is_nan_float, 196
is_nan_int, 196
is_nan_mpq, 196
is_nan_mpz, 197
is_pinf_float, 197
is_pinf_int, 197
is_pinf_mpq, 197
is_pinf_mpz, 197
isqrtrem, 197
lcm_gcd_exact, 198
lcm_mpz, 198
le, 198, 221
le_p, 198
limit_precision, 198
lt, 199, 222
lt_p, 199
mp_size_field_t, 177
mul_2exp_float, 199
mul_2exp_mpq, 199
mul_2exp_mpz, 199
mul_2exp_signed_int, 199
mul_2exp_unsigned_int, 200
mul_float, 200
mul_int_larger, 200
mul_mpq, 200
mul_mpz, 200
mul_signed_int, 200
mul_unsigned_int, 201
neg_float, 201
neg_int_larger, 201
neg_mpq, 201
neg_mpz, 201
neg_signed_int, 201
neg_unsigned_int, 202
nonconst, 222
output_char, 202
output_float, 202
output_int, 202
output_mpq, 202
output_mpz, 202
parse_number, 203
parse_number_part, 203
PPL_DECLARE_FUN1_0_0, 203
PPL_DECLARE_FUN1_0_2, 203
PPL_DECLARE_FUN1_1_2, 204
PPL_DECLARE_FUN2_0_0, 204
PPL_DECLARE_FUN2_0_1, 204, 205
PPL_DECLARE_FUN2_0_2, 205
PPL_DECLARE_FUN3_0_1, 206, 207
PPL_FUNCTION_CLASS, 207
PPL_SAFE_CONVERSION, 208, 209
pred_float, 210
prepare_inexact, 210
rem_float, 210
rem_mpq, 210
rem_mpz, 210
rem_signed_int, 210
rem_unsigned_int, 211
Result, 222
Result_Class, 223
Result_Relation, 223
result_relation, 211
rint, 211
round, 211
round_gt_float, 211
round_gt_int, 212
round_gt_int_no_overflow, 212
round_gt_mpz, 212
round_lt_float, 212
round_lt_int, 212
round_lt_int_no_overflow, 212
round_lt_mpz, 213
set_mp_size, 213
set_neg_overflow_float, 213
set_neg_overflow_int, 213
set_pos_overflow_float, 213
set_pos_overflow_int, 213
sgn_float, 214
sgn_generic, 214
sgn_mp, 214
smod_2exp_float, 214
smod_2exp_mpq, 214
smod_2exp_mpz, 214
smod_2exp_signed_int, 214
smod_2exp_unsigned_int, 215
sqrt_float, 215
sqrt_mpq, 215
sqrt_mpz, 215
sqrt_signed_int, 215
sqrt_unsigned_int, 216
sub_2exp_float, 216
sub_2exp_mpq, 216
sub_2exp_mpz, 216
sub_2exp_signed_int, 216
sub_2exp_unsigned_int, 216
sub_float, 217
sub_int_larger, 217
sub_mpq, 217
sub_mpz, 217
sub_mul_float, 217
sub_mul_int, 217
sub_mul_mpq, 218
sub_mul_mpz, 218
sub_signed_int, 218
sub_unsigned_int, 218
succ_float, 218
sum_sign, 218

To, 223
To1, 224
To2, 224
To3, 224
trunc_float, 219
trunc_mpq, 219
Type, 224
uint_fast16_t, 178
uint_fast32_t, 178
uint_fast64_t, 178
umod_2exp_float, 219
umod_2exp_mpq, 219
umod_2exp_mpz, 219
umod_2exp_signed_int, 219
umod_2exp_unsigned_int, 220
Parma_Polyhedra_Library::Checked::Extended_Int, 648
max, 649
min, 649
minus_infinity, 649
not_a_number, 649
plus_infinity, 650
Parma_Polyhedra_Library::Checked::Float_2exp, 670
const_bool_nodef, 670
Parma_Polyhedra_Library::Checked::Larger< char >, 997
const_bool_nodef, 999
type_for_add, 998
type_for_mul, 998
type_for_neg, 998
type_for_sub, 998
Parma_Polyhedra_Library::Checked::Larger< signed char >, 999
const_bool_nodef, 1000
type_for_add, 1000
type_for_mul, 1000
type_for_neg, 1000
type_for_sub, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1001
const_bool_nodef, 1002
type_for_add, 1001
type_for_mul, 1001
type_for_neg, 1001
type_for_sub, 1001
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1002
const_bool_nodef, 1003, 1004
type_for_add, 1003
type_for_mul, 1003
type_for_neg, 1003
type_for_sub, 1003
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1004
const_bool_nodef, 1005
type_for_add, 1004
type_for_mul, 1004
type_for_neg, 1005
type_for_sub, 1005
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1005
const_bool_nodef, 1007
type_for_add, 1006
type_for_mul, 1006
type_for_neg, 1006
type_for_sub, 1006
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1007
const_bool_nodef, 1008
type_for_add, 1008
type_for_mul, 1008
type_for_neg, 1008
type_for_sub, 1008
Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1009
const_bool_nodef, 1010
type_for_add, 1009
type_for_mul, 1009
type_for_neg, 1009
type_for_sub, 1009
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1010
const_bool_nodef, 1011, 1012
type_for_add, 1011
type_for_mul, 1011
type_for_neg, 1011
type_for_sub, 1011
Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1012
const_bool_nodef, 1013
type_for_add, 1012
type_for_mul, 1012
type_for_neg, 1013
type_for_sub, 1013
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1013
const_bool_nodef, 1015
type_for_add, 1014
type_for_mul, 1014
type_for_neg, 1014
type_for_sub, 1014
Parma_Polyhedra_Library::Checked::number_struct, 1145
base, 1145
base_for_exponent, 1145
exponent, 1145

mantissa, 1146
neg_exponent, 1146
neg_mantissa, 1146
Parma_Polyhedra_Library::Checked::Safe_-Conversion, 1571
Parma_Polyhedra_Library::Checked::Safe_-Conversion< T, T >, 1573
Parma_Polyhedra_Library::Checked::Safe_Int_-Comparison, 1574
Parma_Polyhedra_Library::Checked::Safe_-Int_Comparison< T1, T2, typename Enable_If<(C_Integer< T1 >::value &&C_Integer< T2 >::value)>::type >, 1576
Parma_Polyhedra_Library::Checked_Number, 446
abs_assign, 465
add_mul_assign, 465
assign_r, 466
ceil_assign, 466
Checked_Number, 454–460
classify, 461
cmp, 466
construct, 467
div_2exp_assign, 467
exact_div_assign, 467
external_memory_in_bytes, 467
floor_assign, 467
gcd_assign, 468
gcdext_assign, 468
input, 468
is_infinity, 470
is_integer, 470
is_minus_infinity, 470
is_not_a_number, 470
is_plus_infinity, 470
lcm_assign, 470
mul_2exp_assign, 471
neg_assign, 471
OK, 461
operator T, 461
operator<, 472
operator<<, 472
operator<=, 472
operator>, 472
operator>>, 473
operator>=, 472
operator*=, 462
operator+, 471
operator++, 462
operator+=, 462, 463
operator-, 471
operator--, 463
operator-=, 463
operator/=, 464
operator=, 464
operator==, 472
operator%=, 461, 462
output, 473
raw_value, 465, 473
sgn, 473
sqrt_assign, 474
sub_mul_assign, 474
swap, 474
total_memory_in_bytes, 474
trunc_assign, 474
v, 475
Parma_Polyhedra_Library::Checked_Number_-Transparent_Policy, 475
const_bool_nodef, 477, 478
handle_result, 479
ROUND_DEFAULT_CONSTRUCTOR, 479
ROUND_DEFAULT_FUNCTION, 479
ROUND_DEFAULT_INPUT, 479
ROUND_DEFAULT_OPERATOR, 479
ROUND_DEFAULT_OUTPUT, 479
Parma_Polyhedra_Library::Coefficient_traits_-template, 480
Parma_Polyhedra_Library::Coefficient_traits_-template< GMP_Integer >, 480
const_reference, 481
Parma_Polyhedra_Library::Compile_Time_-Check< true >, 484
Parma_Polyhedra_Library::Congruence, 484
~Congruence, 491
ascii_dump, 492
ascii_load, 492
coefficient, 493
Congruence, 491, 492
create, 493
external_memory_in_bytes, 494
finalize, 494
inhomogeneous_term, 494
initialize, 494
is_equal_at_dimension, 495
is_equality, 495
is_inconsistent, 495
is_proper_congruence, 496
is_tautological, 496
max_space_dimension, 496
modulus, 497
negate, 497
normalize, 497
OK, 498
operator<<, 502
operator/, 501, 502
operator/=, 498
operator=, 498
operator==, 502

operator%=, 501
Parma_Polyhedra_Library::Congruence_-System, 502
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 502
Parma_Polyhedra_Library::Constraint, 502
Parma_Polyhedra_Library::Grid, 503
Parma_Polyhedra_Library::IO_-Operators::operator<<, 503
Parma_Polyhedra_Library::Linear_-Expression, 503
Parma_Polyhedra_Library::Scalar_Products, 503
Parma_Polyhedra_Library::Constraint, 586
Parma_Polyhedra_Library::Linear_-Expression, 1037
print, 498
set_is_equality, 498
sign_normalize, 499
space_dimension, 499
std::swap, 503
strong_normalize, 499
swap, 500, 503
throw_dimension_incompatible, 500
throw_invalid_argument, 500
total_memory_in_bytes, 500
zero_dim_false, 500
zero_dim_false_p, 503
zero_dim_integrality, 501
zero_dim_integrality_p, 504
Parma_Polyhedra_Library::Congruence_System, 504
∼Congruence_System, 510
add_unit_rows_and_columns, 510
affine_preimage, 511
ascii_dump, 511, 512
ascii_load, 512
begin, 512
clear, 513
concatenate, 513
Congruence_System, 509, 510
const_iterator, 521
empty, 513
end, 513
external_memory_in_bytes, 514
finalize, 514
Grid, 521
Grid_Certificate, 522
has_a_free_dimension, 514
has_linear_equalities, 514
increase_space_dimension, 515
initialize, 515
insert, 515, 516
insert_verbatim, 516
is_equal_to, 517
max_space_dimension, 517
normalize_moduli, 517
num_equalities, 517
num_proper_congruences, 518
OK, 518
operator<<, 522
operator=, 518
operator==, 522
Parma_Polyhedra_Library::Congruence, 502
Parma_Polyhedra_Library::Linear_-Expression, 1037
print, 519
recycling_insert, 519
remove_higher_space_dimensions, 519
resize_no_copy, 519
satisfies_all_congruences, 520
space_dimension, 520
std::swap, 522
swap, 521, 522
total_memory_in_bytes, 521
zero_dim_empty, 521
zero_dim_empty_p, 522
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 539
∼const_iterator, 541
Congruence_System, 543
const_iterator, 541
csp, 543
i, 543
operator∗, 541
operator++, 542
operator->, 542
operator=, 542
operator==, 542
Parma_Polyhedra_Library::Congruence, 502
skip_forward, 543
Parma_Polyhedra_Library::Congruences_-Reduction, 523
∼Congruences_Reduction, 524
Congruences_Reduction, 524
product_reduce, 524
Parma_Polyhedra_Library::const_iterator_to_-const, 554
Base, 556
base, 560
const_iterator_to_const, 557
difference_type, 556
iterator_category, 556
iterator_to_const< Container >, 559
operator∗, 558
operator++, 558
operator->, 559
operator--, 558, 559

operator==, 559, 560
pointer, 556
Powerset, 560
reference, 556
Traits, 557
value_type, 557
Parma_Polyhedra_Library::Constant, 560
Parma_Polyhedra_Library::Constant_, 561
Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_-If<(Fit< typename C_Integer< T >::smaller_signed_type, v >::value &&(prefer_signed, $562
Parma_Polyhedra_Library::Constant_< T, v, prefer_signed, typename Enable_-If<(Fit< typename C_Integer< T >::smaller_unsigned_type, v >::value &&(
prefer_signed, $562
Parma_Polyhedra_Library::Constraint, 563
~Constraint, 571
ascii_dump, 572
ascii_load, 572
coefficient, 573
Constraint, 571, 572
construct_epsilon_geq_zero, 573
epsilon_geq_zero, 573
epsilon_geq_zero_p, 587
epsilon_leq_one, 574
epsilon_leq_one_p, 587
external_memory_in_bytes, 574
finalize, 574
inhomogeneous_term, 574
initialize, 575
is_equality, 575
is_equivalent_to, 576
is_inconsistent, 576
is_inequality, 576
is_nonstrict_inequality, 577
is_strict_inequality, 577
is_tautological, 577
max_space_dimension, 578
OK, 578
operator<, 582
operator<<, 583
operator<=, 583
operator>, 584, 585
operator>=, 585, 586
operator=, 578
operator==, 584
Parma_Polyhedra_Library::Congruence, 586
Parma_Polyhedra_Library::Constraint_-System, 586
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 586
Parma_Polyhedra_Library::Linear_-Expression::Linear_Expression, 579
Parma_Polyhedra_Library::Polyhedron, 586
Parma_Polyhedra_Library::Scalar_Products, 586
Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Sign, 586
Parma_Polyhedra_Library::Congruence, 502
Parma_Polyhedra_Library::Linear_-Expression, 1037
Parma_Polyhedra_Library::Linear_Row, 1055
print, 579
set_is_equality, 579
set_is_inequality, 579
space_dimension, 579
swap, 580, 586
throw_dimension_incompatible, 580
throw_invalid_argument, 580
total_memory_in_bytes, 581
Type, 571
type, 581
zero_dim_false, 581
zero_dim_false_p, 587
zero_dim_positivity, 581
zero_dim_positivity_p, 587
Parma_Polyhedra_Library::Constraint_System, 588
~Constraint_System, 593
add_low_level_constraints, 594
adjust_topology_and_space_dimension, 594
affine_preimage, 595
ascii_dump, 595, 596
ascii_load, 596
begin, 596
clear, 597
const_iterator, 605
Constraint_System, 593, 594
empty, 597
end, 598
external_memory_in_bytes, 598
finalize, 599
has_equalities, 599
has_strict_inequalities, 599
initialize, 599
insert, 600
insert_pending, 600
max_space_dimension, 601
num_equalities, 601
num_inequalities, 601
OK, 601
operator<<, 605
operator=, 602
operator==, 605

Parma_Polyhedra_Library::Polyhedron, 605
Parma_Polyhedra_Library::Constraint, 586
Parma_Polyhedra_Library::Linear_-Expression, 1037
print, 602
satisfies_all_constraints, 602
simplify, 603
space_dimension, 603
swap, 604, 605
total_memory_in_bytes, 604
zero_dim_empty, 604
zero_dim_empty_p, 606
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 534
∼const_iterator, 536
const_iterator, 536
Constraint_System, 538
csp, 538
i, 538
operator∗, 537
operator++, 537
operator->, 537
operator=, 537
operator==, 538
Parma_Polyhedra_Library::Constraint, 586
skip_forward, 538
Parma_Polyhedra_Library::Constraints_Reduction, 606
∼Constraints_Reduction, 607
Constraints_Reduction, 607
product_reduce, 607
Parma_Polyhedra_Library::DB_Matrix, 608
∼DB_Matrix, 612
ascii_dump, 612
ascii_load, 612
begin, 612
DB_Matrix, 611, 616
end, 613
euclidean_distance_assign, 616
external_memory_in_bytes, 613
grow, 613
l_infinity_distance_assign, 617
l_m_distance_assign, 617
max_num_columns, 614
max_num_rows, 614
num_rows, 614
OK, 614
operator<<, 617
operator=, 615
operator==, 618
print, 615
rectilinear_distance_assign, 618
resize_no_copy, 615
row_capacity, 618
row_size, 619
rows, 619
swap, 616, 618
total_memory_in_bytes, 616
Parma_Polyhedra_Library::DB_Matrix::const_-iterator, 529
const_iterator, 532
difference_type, 531
i, 534
Iter, 531
iterator_category, 531
operator∗, 532
operator++, 532, 533
operator->, 533
operator=, 533
operator==, 533
pointer, 531
reference, 531
value_type, 531
Parma_Polyhedra_Library::DB_Row, 619
∼DB_Row, 625
allocate, 626
assign, 626
begin, 626
const_iterator, 624
construct, 627
construct_upward_approximation, 627
copy_construct_coefficients, 628
DB_Row, 624, 625
end, 628
expand_within_capacity, 628
external_memory_in_bytes, 629
iter_swap, 631
iterator, 624
max_size, 629
OK, 629
operator=, 629
operator==, 632
Parma_Polyhedra_Library::DB_Row, 632
Parma_Polyhedra_Library::DB_Row, 632
shrink, 630
size, 630
swap, 631, 632
total_memory_in_bytes, 631
Parma_Polyhedra_Library::DB_Row_Impl_-Handler, 632
∼DB_Row_Impl_Handler, 634
DB_Row_Impl_Handler, 634
impl, 634
operator=, 634
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 890
∼Impl, 892
bump_size, 893

construct_upward_approximation, 893
copy_construct, 893
copy_construct_coefficients, 893
DB_Row< T >, 897
expand_within_capacity, 894
external_memory_in_bytes, 894
Impl, 892, 893
max_size, 894
operator delete, 895
operator new, 895
operator=, 895
set_size, 896
shrink, 896
size, 896
size_, 897
total_memory_in_bytes, 896, 897
vec_, 897
Parma_Polyhedra_Library::Determinate, 635
~Determinate, 639
concatenate_assign, 639
definitely_entails, 639
Determinate, 638
external_memory_in_bytes, 639
has_nontrivial_weakening, 640
is_bottom, 640
is_definitely_equivalent_to, 640
is_top, 640
lift_op_assign, 640
meet_assign, 641
mutate, 641
OK, 641
operator<<, 643
operator=, 641
operator==, 643
pointset, 642
prep, 644
swap, 642, 644
total_memory_in_bytes, 642
upper_bound_assign, 643
weakening_assign, 643
Parma_Polyhedra_Library::Determinate::Binary_-Operator_Assign_Lifter, 342
Binary_Operator_Assign_Lifter, 343
op_assign_, 343
operator(), 343
Parma_Polyhedra_Library::Determinate::Rep, 1542
~Rep, 1544
del_reference, 1544
external_memory_in_bytes, 1545
is_shared, 1545
new_reference, 1545
operator=, 1545
pset, 1546
references, 1546
Rep, 1543, 1544
total_memory_in_bytes, 1545
Parma_Polyhedra_Library::Domain_Product, 644
Congruences_Product, 645
Constraints_Product, 645
Direct_Product, 645
Shape_Preserving_Product, 645
Smash_Product, 645
Parma_Polyhedra_Library::Enable_If, 646
Parma_Polyhedra_Library::Enable_If< true, T >, 646
type, 647
Parma_Polyhedra_Library::Enable_If_Is, 647
type, 647
Parma_Polyhedra_Library::euclidean_distance_-assign
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Octagonal_Shape, 1223
Parma_Polyhedra_Library::Euclidean_Distance_-Specialization, 648
combine, 648
finalize, 648
Parma_Polyhedra_Library::Extended_Number_-Policy, 650
const_bool_nodef, 651, 652
handle_result, 652
Parma_Polyhedra_Library::False, 652
Parma_Polyhedra_Library::Fit, 653
Parma_Polyhedra_Library::Fit< T, v, typename Enable_If< C_Integer< T >::value >::type >, 655
Parma_Polyhedra_Library::Float, 668
Parma_Polyhedra_Library::float_ibm_double, 670
BASE, 671
EXPONENT_BIAS, 671
EXPONENT_BITS, 671
MANTISSA_BITS, 671
Parma_Polyhedra_Library::float_ibm_single, 671
BASE, 673
build, 672
dec, 672
EXP_MASK, 673
EXPONENT_BIAS, 674
EXPONENT_BITS, 674
EXPONENT_MAX, 674
EXPONENT_MIN, 674
EXPONENT_MIN_DENORM, 674
inc, 672
is_inf, 672
is_nan, 673
is_zero, 673
MANTISSA_BITS, 674
NEG_INF, 674

NEG_ZERO, 675
negate, 673
POS_INF, 675
POS_ZERO, 675
set_max, 673
SGN_MASK, 675
sign_bit, 673
word, 675
Parma_Polyhedra_Library::float_ieee754_double, 676
BASE, 678
build, 677
dec, 677
EXPONENT_BIAS, 678
EXPONENT_BITS, 678
EXPONENT_MAX, 678
EXPONENT_MIN, 678
EXPONENT_MIN_DENORM, 679
inc, 677
is_inf, 677
is_nan, 677
is_zero, 677
lsp, 679
LSP_INF, 679
LSP_MAX, 679
LSP_ZERO, 679
MANTISSA_BITS, 679
msp, 680
MSP_NEG_INF, 680
MSP_NEG_ZERO, 680
MSP_POS_INF, 680
MSP_POS_ZERO, 680
MSP_SGN_MASK, 680
negate, 677
set_max, 678
sign_bit, 678
Parma_Polyhedra_Library::float_ieee754_half, 681
BASE, 683
build, 682
dec, 682
EXP_MASK, 683
EXPONENT_BIAS, 683
EXPONENT_BITS, 683
EXPONENT_MAX, 684
EXPONENT_MIN, 684
EXPONENT_MIN_DENORM, 684
inc, 682
is_inf, 682
is_nan, 682
is_zero, 682
MANTISSA_BITS, 684
NEG_INF, 684
NEG_ZERO, 684
negate, 682
POS_INF, 684
POS_ZERO, 685
set_max, 683
SGN_MASK, 685
sign_bit, 683
word, 685
Parma_Polyhedra_Library::float_ieee754_quad, 685
BASE, 687
build, 686
dec, 686
EXPONENT_BIAS, 688
EXPONENT_BITS, 688
EXPONENT_MAX, 688
EXPONENT_MIN, 688
EXPONENT_MIN_DENORM, 688
inc, 686
is_inf, 686
is_nan, 687
is_zero, 687
lsp, 688
LSP_INF, 688
LSP_MAX, 689
LSP_ZERO, 689
MANTISSA_BITS, 689
msp, 689
MSP_NEG_INF, 689
MSP_NEG_ZERO, 689
MSP_POS_INF, 690
MSP_POS_ZERO, 690
MSP_SGN_MASK, 690
negate, 687
set_max, 687
sign_bit, 687
Parma_Polyhedra_Library::float_ieee754_single, 690
BASE, 693
build, 691
dec, 691
EXP_MASK, 693
EXPONENT_BIAS, 693
EXPONENT_BITS, 693
EXPONENT_MAX, 693
EXPONENT_MIN, 693
EXPONENT_MIN_DENORM, 693
inc, 691
is_inf, 692
is_nan, 692
is_zero, 692
MANTISSA_BITS, 694
NEG_INF, 694
NEG_ZERO, 694
negate, 692
POS_INF, 694

POS_ZERO, 694
set_max, 692
SGN_MASK, 694
sign_bit, 692
word, 694
Parma_Polyhedra_Library::float_intel_double_-extended, 695
BASE, 697
build, 696
dec, 696
EXPONENT_BIAS, 697
EXPONENT_BITS, 697
EXPONENT_MAX, 697
EXPONENT_MIN, 698
EXPONENT_MIN_DENORM, 698
inc, 696
is_inf, 696
is_nan, 696
is_zero, 696
lsp, 698
LSP_DMAX, 698
LSP_INF, 698
LSP_NMAX, 698
LSP_ZERO, 698
MANTISSA_BITS, 699
msp, 699
MSP_NEG_INF, 699
MSP_NEG_ZERO, 699
MSP_POS_INF, 699
MSP_POS_ZERO, 699
MSP_SGN_MASK, 700
negate, 696
set_max, 697
sign_bit, 697
Parma_Polyhedra_Library::FPU_Related< Checked_Number< T, Policy > >, 700
Parma_Polyhedra_Library::Generator, 700
~Generator, 709
ascii_dump, 710
ascii_load, 710
closure_point, 711
coefficient, 711
divisor, 712
euclidean_distance_assign, 720
external_memory_in_bytes, 712
finalize, 712
Generator, 709, 710
initialize, 713
is_closure_point, 713
is_equivalent_to, 713
is_line, 713
is_line_or_ray, 714
is_matching_closure_point, 714
is_point, 714
is_ray, 715
is_ray_or_point, 715
l_infinity_distance_assign, 721
l_m_distance_assign, 722
line, 715
max_space_dimension, 715
OK, 716
operator<<, 722
operator=, 716
operator==, 722
Parma_Polyhedra_Library::Generator_-System, 723
Parma_Polyhedra_Library::Generator_-System::const_iterator, 723
Parma_Polyhedra_Library::Grid_Generator, 723
Parma_Polyhedra_Library::Grid_Generator_-System, 723
Parma_Polyhedra_Library::IO_-Operators::operator<<, 723
Parma_Polyhedra_Library::Linear_-Expression::Linear_Expression, 716
Parma_Polyhedra_Library::Polyhedron, 723
Parma_Polyhedra_Library::Scalar_Products, 723
Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Assign, 723
Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Sign, 724
Parma_Polyhedra_Library::Linear_-Expression, 1037
Parma_Polyhedra_Library::Linear_Row, 1055
point, 716
print, 717
ray, 717
rectilinear_distance_assign, 724
set_is_line, 717
set_is_ray_or_point, 717
space_dimension, 718
swap, 718, 725
throw_dimension_incompatible, 718
throw_invalid_argument, 719
total_memory_in_bytes, 719
Type, 709
type, 719
zero_dim_closure_point, 719
zero_dim_closure_point_p, 725
zero_dim_point, 720
zero_dim_point_p, 725
Parma_Polyhedra_Library::Generator_System, 726
~Generator_System, 732
add_corresponding_closure_points, 733
add_corresponding_points, 733

adjust_topology_and_space_dimension, 733
affine_image, 734
ascii_dump, 735
ascii_load, 735
begin, 736
clear, 736
const_iterator, 744
empty, 736
end, 736
external_memory_in_bytes, 737
finalize, 737
Generator_System, 732, 733
has_closure_points, 737
has_points, 737
initialize, 738
insert, 738
insert_pending, 738
max_space_dimension, 739
num_lines, 739
num_rays, 739
OK, 740
operator<<, 744
operator=, 740
operator==, 744
Parma_Polyhedra_Library::Grid_Generator_-System, 744
Parma_Polyhedra_Library::Polyhedron, 744
Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Linear_-Expression, 1037
print, 741
relation_with, 741
remove_invalid_lines_and_rays, 741
satisfied_by_all_generators, 741
satisfied_by_all_generators_C, 742
satisfied_by_all_generators_NNC, 742
simplify, 742
space_dimension, 742
swap, 743, 744
total_memory_in_bytes, 743
zero_dim_univ, 743
zero_dim_univ_p, 744
Parma_Polyhedra_Library::Generator_-System::const_iterator, 544
∼const_iterator, 546
const_iterator, 546
Generator_System, 548
gsp, 549
i, 549
operator∗, 547
operator++, 547
operator->, 547
operator=, 548
operator==, 548
Parma_Polyhedra_Library::Generator, 723
skip_forward, 548
Parma_Polyhedra_Library::GMP_Integer, 745
abs_assign, 746
add_mul_assign, 746
cmp, 747
div_2exp_assign, 747
exact_div_assign, 747
gcd_assign, 747
gcdext_assign, 747
lcm_assign, 747
mul_2exp_assign, 748
neg_assign, 748
raw_value, 748
rem_assign, 748
sqrt_assign, 749
sub_mul_assign, 749
Parma_Polyhedra_Library::Grid, 749
∼Grid, 771
add_congruence, 772
add_congruence_no_check, 772
add_congruences, 772
add_constraint, 773
add_constraint_no_check, 773
add_constraints, 773
add_grid_generator, 774
add_grid_generators, 774
add_recycled_congruences, 775
add_recycled_constraints, 775
add_recycled_grid_generators, 776
add_space_dimensions, 776, 777
add_space_dimensions_and_embed, 777
add_space_dimensions_and_project, 778
affine_dimension, 778
affine_image, 778
affine_preimage, 779
ascii_dump, 780, 781
ascii_load, 781
bounded_affine_image, 781
bounded_affine_preimage, 782
bounds, 782
bounds_from_above, 783
bounds_from_below, 783
bounds_no_check, 783
can_recycle_congruence_systems, 784
can_recycle_constraint_systems, 784
clear_congruences_minimized, 784
clear_congruences_up_to_date, 784
clear_empty, 784
clear_generators_minimized, 785
clear_generators_up_to_date, 785
coefficient_type, 765
con_sys, 830
concatenate_assign, 785

congruence_widening_assign, 785
congruences, 786
congruences_are_minimized, 786
congruences_are_up_to_date, 787
constrains, 787
constraints, 787
construct, 787, 788
contains, 789
contains_integer_point, 789
conversion, 789
difference_assign, 790
dim_kinds, 830
Dimension_Kind, 766
Dimension_Kinds, 765
drop_some_non_integer_points, 790, 791
expand_space_dimension, 791
external_memory_in_bytes, 791
fold_space_dimensions, 792
frequency, 792
frequency_no_check, 793
gen_sys, 830
generalized_affine_image, 793, 794
generalized_affine_preimage, 795
generator_widening_assign, 796
generators_are_minimized, 796
generators_are_up_to_date, 797
Grid, 766–771
grid_generators, 797
hash_code, 797
intersection_assign, 798
is_bounded, 798
is_discrete, 798
is_disjoint_from, 798
is_empty, 799
is_included_in, 799
is_topologically_closed, 799
is_universe, 799
limited_congruence_extrapolation_assign, 800
limited_extrapolation_assign, 800
limited_generator_extrapolation_assign, 800
lower_triangular, 801
map_space_dimensions, 801
marked_empty, 802
max_min, 803
max_space_dimension, 803
maximize, 804
minimize, 805, 806
minimized_congruences, 806
minimized_constraints, 806
minimized_grid_generators, 807
multiply_grid, 807
normalize_divisors, 807, 808
OK, 809
operator<<, 829
operator=, 809
operator==, 829
Parma_Polyhedra_Library::Box, 829
Parma_Polyhedra_Library::Grid_Certificate, 830
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Linear_-Expression, 1037
Parma_Polyhedra_Library::Polyhedron, 1510
print, 810
quick_equivalence_test, 810
reduce_congruence_with_equality, 810
reduce_equality_with_equality, 810
reduce_line_with_line, 811
reduce_parameter_with_line, 811
reduce_pc_with_pc, 811
reduce_reduced, 812
refine_no_check, 812
refine_with_congruence, 813
refine_with_congruences, 813
refine_with_constraint, 813
refine_with_constraints, 814
relation_with, 814, 815
remove_higher_space_dimensions, 815
remove_space_dimensions, 816
rows_are_zero, 816
select_wider_congruences, 817
select_wider_generators, 817
set_congruences_minimized, 817
set_congruences_up_to_date, 817
set_empty, 818
set_generators_minimized, 818
set_generators_up_to_date, 818
set_zero_dim_univ, 819
simplify, 819
simplify_using_context_assign, 820
space_dim, 831
space_dimension, 820
status, 831
strictly_contains, 820
swap, 821, 830
Three_Valued_Boolean, 766
throw_dimension_incompatible, 821–823
throw_invalid_argument, 823
throw_invalid_constraint, 823
throw_invalid_constraints, 824
throw_invalid_generator, 824
throw_invalid_generators, 824
throw_runtime_error, 824
throw_space_dimension_overflow, 824
time_elapse_assign, 824
topological_closure_assign, 825
total_memory_in_bytes, 825
unconstrain, 825

update_congruences, 826
update_generators, 826
upper_bound_assign, 827
upper_bound_assign_if_exact, 827
upper_triangular, 827
widening_assign, 827
wrap_assign, 828
Parma_Polyhedra_Library::Grid::Status, 1600
ascii_dump, 1604
ascii_load, 1604
C_MINIMIZED, 1608
C_UP_TO_DATE, 1608
CS_PENDING, 1608
EMPTY, 1608
flags, 1608
flags_t, 1603
G_MINIMIZED, 1609
G_UP_TO_DATE, 1609
GS_PENDING, 1609
OK, 1604
print, 1604
reset, 1604
reset_c_minimized, 1604
reset_c_pending, 1604
reset_c_up_to_date, 1605
reset_empty, 1605
reset_g_minimized, 1605
reset_g_pending, 1605
reset_g_up_to_date, 1605
reset_sat_c_up_to_date, 1605
reset_sat_g_up_to_date, 1605
reset_zero_dim_univ, 1605
SAT_C_UP_TO_DATE, 1609
SAT_G_UP_TO_DATE, 1609
set, 1605
set_c_minimized, 1605
set_c_pending, 1606
set_c_up_to_date, 1606
set_empty, 1606
set_g_minimized, 1606
set_g_pending, 1606
set_g_up_to_date, 1606
set_sat_c_up_to_date, 1606
set_sat_g_up_to_date, 1606
set_zero_dim_univ, 1606
Status, 1603
test_all, 1607
test_any, 1607
test_c_minimized, 1607
test_c_pending, 1607
test_c_up_to_date, 1607
test_empty, 1607
test_g_minimized, 1607
test_g_pending, 1607
test_g_up_to_date, 1607
test_sat_c_up_to_date, 1608
test_sat_g_up_to_date, 1608
test_zero_dim_univ, 1608
ZERO_DIM_UNIV, 1609
Parma_Polyhedra_Library::Grid_Certificate, 832
~Grid_Certificate, 834
compare, 834
Grid_Certificate, 833
is_stabilizing, 834
num_equalities, 835
num_proper_congruences, 835
OK, 834
Parma_Polyhedra_Library::Grid, 830
Parma_Polyhedra_Library::Grid_-
Certificate::Compare, 482
operator(), 483
Parma_Polyhedra_Library::Grid_Generator, 835
~Grid_Generator, 844
all_homogeneous_terms_are_zero, 844
ascii_dump, 844, 845
ascii_load, 845
coefficient, 845
coefficient_swap, 845
Congruence_System, 855
divisor, 846
external_memory_in_bytes, 846
finalize, 846
Grid, 855
Grid_Generator, 844
Grid_Generator_System, 855
Grid_Generator_System::const_iterator, 855
grid_line, 847
grid_point, 847
initialize, 847
IO_Operators::operator<<, 855
is_equal_at_dimension, 847
is_equal_to, 848
is_equivalent_to, 848
is_line, 848
is_line_or_parameter, 848
is_parameter, 849
is_parameter_or_point, 849
is_point, 849
Linear_Expression, 855
max_space_dimension, 849
negate, 849
OK, 850
operator<<, 855, 856
operator=, 850
operator==, 856
parameter, 851
Parma_Polyhedra_Library::Generator, 723

Parma_Polyhedra_Library::Linear_-
Expression, 1037
print, 851
Scalar_Products, 856
scale_to_divisor, 851
set_divisor, 852
set_is_line, 852
set_is_parameter, 852
set_is_parameter_or_point, 852
size, 853
space_dimension, 853
strong_normalize, 853
swap, 853, 856
throw_invalid_argument, 854
Topology_Adjusted_Scalar_Product_Sign,
856
total_memory_in_bytes, 854
Type, 843
type, 854
zero_dim_point, 854
zero_dim_point_p, 857
Parma_Polyhedra_Library::Grid_Generator_-
System, 857
∼Grid_Generator_System, 864
add_universe_rows_and_columns, 864
affine_image, 864
ascii_dump, 865
ascii_load, 865
begin, 866
clear, 866
empty, 866
end, 866
erase_to_end, 867
external_memory_in_bytes, 867
finalize, 867
Grid, 875
Grid_Generator_System, 863, 864
has_points, 867
initialize, 867
insert, 868
is_equal_to, 868
max_space_dimension, 868
num_columns, 869
num_lines, 869
num_parameters, 869
num_rows, 869
OK, 870
operator<<, 875
operator=, 870
operator==, 875
Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Generator_-
System, 744
Parma_Polyhedra_Library::Linear_-
Expression, 1038
permute_columns, 871
print, 871
recycling_insert, 871, 872
remove_higher_space_dimensions, 872
remove_space_dimensions, 872
resize_no_copy, 873
set_index_first_pending_row, 873
set_sorted, 873
space_dimension, 873
swap, 874, 875
total_memory_in_bytes, 874
unset_pending_rows, 874
zero_dim_univ, 874
zero_dim_univ_p, 876
Parma_Polyhedra_Library::Grid_Generator_-
System::const_iterator, 549
∼const_iterator, 552
const_iterator, 551, 552
Grid_Generator_System, 554
operator∗, 552
operator++, 552, 553
operator->, 553
operator=, 553
operator==, 553
Parma_Polyhedra_Library::H79_Certificate, 876
∼H79_Certificate, 878
affine_dim, 879
compare, 878
H79_Certificate, 877, 878
num_constraints, 879
Parma_Polyhedra_Library::Polyhedron, 1510
Parma_Polyhedra_Library::H79_-
Certificate::Compare, 482
operator(), 482
Parma_Polyhedra_Library::Has_OK, 879
Parma_Polyhedra_Library::Has_OK< T, type-
name Enable_If_Is< bool(T::∗)()
const,&T::OK >::type >, 881
Parma_Polyhedra_Library::Implementation, 225
first_one, 225
in_assert, 226
last_one, 225
Parma_Polyhedra_-
Library::Implementation::Octagonal_-
Shapes, 226
Parma_Polyhedra_-
Library::Implementation::Pointset_-
Powersets, 226
Parma_Polyhedra_-
Library::Implementation::Termination,
226
all_affine_quasi_ranking_functions_MS, 227

all_affine_ranking_functions_MS, 227
all_affine_ranking_functions_PR, 227
all_affine_ranking_functions_PR_original, 227
assign_all_inequalities_approximation, 227, 228
fill_constraint_system_PR, 228
fill_constraint_system_PR_original, 229
fill_constraint_systems_MS, 230
one_affine_ranking_function_MS, 230
one_affine_ranking_function_PR, 230
one_affine_ranking_function_PR_original, 231
shift_unprimed_variables, 231
termination_test_MS, 231
termination_test_PR, 231
termination_test_PR_original, 231
Parma_Polyhedra_Library::Init, 898
~Init, 899
count, 900
Init, 899
old_rounding_direction, 900
restore_pre_PPL_rounding, 899
set_rounding_for_PPL, 899
Parma_Polyhedra_Library::Interfaces, 231
is_necessarily_closed_for_interfaces, 232
Parma_Polyhedra_Library::Interfaces::C, 232
Parma_Polyhedra_Library::Interfaces::is_necessarily_closed_for_interfaces
Parma_Polyhedra_Library::Polyhedron, 1510
Parma_Polyhedra_Library::Interfaces::Java, 232
Parma_Polyhedra_Library::Interfaces::OCaml, 233
Parma_Polyhedra_Library::Interfaces::Prolog, 233
Parma_Polyhedra_Library::Interfaces::Prolog::Ciao, 233
Parma_Polyhedra_Library::Interfaces::Prolog::GNU, 234
Parma_Polyhedra_Library::Interfaces::Prolog::SICStus, 234
Parma_Polyhedra_Library::Interfaces::Prolog::SWI, 234
Parma_Polyhedra_Library::Interfaces::Prolog::XSB, 234
Parma_Polyhedra_Library::Interfaces::Prolog::YAP, 234
Parma_Polyhedra_Library::Interval, 900
add_assign, 907
add_constraint, 907
ascii_dump, 907
ascii_load, 907
assign, 908
boundary_type, 906
build, 908, 909
can_be_exactly_joined_to, 909
CC76_widening_assign, 909
check_empty, 910
contains, 910
contains_integer_point, 910
difference_assign, 910, 911
div_assign, 911
drop_some_non_integer_points, 911
empty_intersection_assign, 912
external_memory_in_bytes, 912
has_restriction, 912
info, 912, 913
info_type, 906
intersect_assign, 913, 914
Interval, 906
is_always_topologically_closed, 914
is_bounded, 914
is_disjoint_from, 914
is_empty, 915
is_infinity, 915
is_singleton, 915
is_topologically_closed, 916
is_universe, 916
join_assign, 916, 917
lower, 917
lower_, 929
lower_approximation_difference_assign, 918
lower_constraint, 918
lower_extend, 918
lower_is_boundary_infinity, 918
lower_is_domain_inf, 919
lower_is_open, 919
lower_normalize, 919
mul_assign, 920
neg_assign, 920
normalize, 920
OK, 921
operator*=, 921
operator+=, 921
operator-=, 922
operator/=, 922
operator=, 922
PPL_COMPILE_TIME_CHECK, 922
Property, 906
refine_existential, 922
refine_universal, 923
remove_inf, 924
remove_sup, 924
set_infinities, 924
simplify_using_context_assign, 924
strictly_contains, 925
sub_assign, 925
swap, 925, 929
topological_closure_assign, 925

total_memory_in_bytes, 926
upper, 926
upper_, 929
upper_constraint, 927
upper_extend, 927
upper_is_boundary_infinity, 927
upper_is_domain_sup, 928
upper_is_open, 928
upper_normalize, 928
w_info, 928
wrap_assign, 929
Parma_Polyhedra_Library::Interval_Base, 930
Parma_Polyhedra_Library::Interval_Info_Bitset, 930
ascii_dump, 932
ascii_load, 932
bitset, 937
clear, 932
clear_boundary_properties, 933
const_bool_nodef, 933, 934
const_int_nodef, 934, 935
get_boundary_property, 935
get_interval_property, 935
Interval_Info_Bitset, 932
set_boundary_property, 936
set_interval_property, 936
swap, 936
Parma_Polyhedra_Library::Interval_Info_Null, 937
ascii_dump, 938
ascii_load, 938
clear, 938
clear_boundary_properties, 938
const_bool_nodef, 939
get_boundary_property, 939
get_interval_property, 940
set_boundary_property, 940
set_interval_property, 940
swap, 940
Parma_Polyhedra_Library::Interval_Info_Null_Open, 941
ascii_dump, 942
ascii_load, 942
const_bool_nodef, 942
get_boundary_property, 942
Interval_Info_Null_Open, 942
open, 943
Parma_Polyhedra_Library::Interval_NS, 235
CARDINALITY_0, 239
CARDINALITY_1, 239
CARDINALITY_IS, 240
f_info, 236
f_is_empty, 237
f_is_singleton, 237
f_lower, 237
f_upper, 238
get_bit, 238
get_bits, 238
reset_bit, 239
reset_bits, 239
Scalar_As_Interval_Info, 236
Scalar_As_Interval_Info_Open, 236
SCALAR_INFO, 240
set_bit, 239
set_bits, 239
Parma_Polyhedra_Library::Interval_NS::Property, 1534
default_value, 1536
Property, 1535
Type, 1535
type, 1536
unsupported_value, 1536
Value, 1535
Parma_Polyhedra_Library::Interval_NS::Scalar_As_Interval_Policy, 1576
const_bool_nodef, 1577
Parma_Polyhedra_Library::Interval_Restriction_Integer, 943
const_int_nodef, 944
get_integer, 944
has_restriction, 944
Interval_Restriction_Integer, 944
normalize, 945
restrict, 945
set_integer, 945
Parma_Polyhedra_Library::Interval_Restriction_Integer_Base, 945
Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo, 946
assign_or_swap, 947
clear, 947
divisor, 949
has_restriction, 948
Interval_Restriction_Integer_Modulo, 947
modulo_type, 947
normalize, 948
PPL_COMPILE_TIME_CHECK, 948
remainder, 949
restrict, 948
Parma_Polyhedra_Library::Interval_Restriction_Integer_Modulo_Base, 949
Parma_Polyhedra_Library::Interval_Restriction_None, 950
Interval_Restriction_None, 950
Parma_Polyhedra_Library::Interval_Restriction_None_Base, 951
has_restriction, 951
normalize, 951
restrict, 952

Parma_Polyhedra_Library::IO_Operators, 240
operator<<, 240
Parma_Polyhedra_Library::IO_-Operators::operator<<
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Box, 434
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Linear_-Expression, 1038
Parma_Polyhedra_Library::Octagonal_Shape, 1223
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1287
Parma_Polyhedra_Library::Poly_Con_-Relation, 1406
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1413
Parma_Polyhedra_Library::Variable, 1670
Parma_Polyhedra_Library::Is_Checked, 952
Parma_Polyhedra_Library::Is_Checked< Checked_Number< T, P > >, 953
Parma_Polyhedra_Library::Is_Interval, 955
Parma_Polyhedra_Library::Is_Native, 956
Parma_Polyhedra_Library::Is_Native< char >, 957
Parma_Polyhedra_Library::Is_Native< mpq_class >, 959
Parma_Polyhedra_Library::Is_Native< mpz_class >, 960
Parma_Polyhedra_Library::Is_Native< signed char >, 962
Parma_Polyhedra_Library::Is_Native< signed int >, 963
Parma_Polyhedra_Library::Is_Native< signed long >, 965
Parma_Polyhedra_Library::Is_Native< signed long long >, 966
Parma_Polyhedra_Library::Is_Native< signed short >, 968
Parma_Polyhedra_Library::Is_Native< unsigned char >, 969
Parma_Polyhedra_Library::Is_Native< unsigned int >, 971
Parma_Polyhedra_Library::Is_Native< unsigned long >, 972
Parma_Polyhedra_Library::Is_Native< unsigned long long >, 974
Parma_Polyhedra_Library::Is_Native< unsigned short >, 975
Parma_Polyhedra_Library::Is_Native_Or_-Checked, 977
Parma_Polyhedra_Library::Is_Same, 978
Parma_Polyhedra_Library::Is_Same< T, T >, 979
Parma_Polyhedra_Library::Is_Same_Or_Derived, 981
anonymous_enum, 982
derived_object, 983
func, 983
PPL_COMPILE_TIME_CHECK, 983
Parma_Polyhedra_Library::Is_Same_Or_-Derived::Any, 241
Any, 242
Parma_Polyhedra_Library::Is_Singleton, 983
Parma_Polyhedra_Library::Is_Special, 985
Parma_Polyhedra_Library::Is_Special< Minus_-Infinity >, 986
Parma_Polyhedra_Library::Is_Special< Not_A_-Number >, 988
Parma_Polyhedra_Library::Is_Special< Plus_-Infinity >, 989
Parma_Polyhedra_Library::iterator_to_const, 991
Base, 993
base, 996
const_iterator_to_const< Container >, 996
difference_type, 993
iterator_category, 993
iterator_to_const, 994
operator*, 994
operator++, 995
operator->, 995
operator--, 995
operator==, 996
pointer, 993
Powerset, 996
reference, 993
Traits, 993
value_type, 993
Parma_Polyhedra_Library::l_infinity_distance_-assign
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Octagonal_Shape, 1223
Parma_Polyhedra_Library::L_Infinity_Distance_-Specialization, 997
combine, 997
finalize, 997
Parma_Polyhedra_Library::l_m_distance_assign
Parma_Polyhedra_Library::Box, 434
Parma_Polyhedra_Library::Limited_Widening_-Function, 1015
Limited_Widening_Function, 1016
Limited_Widening_Method, 1016
limiting_cs, 1017
lw_method, 1017
operator(), 1016
Parma_Polyhedra_Library::Linear_Expression, 1017

~Linear_Expression, 1025
add_mul_assign, 1032
all_homogeneous_terms_are_zero, 1027
ascii_dump, 1027
ascii_load, 1028
coefficient, 1028
external_memory_in_bytes, 1029
finalize, 1029
inhomogeneous_term, 1029
initialize, 1030
is_zero, 1030
Linear_Expression, 1024–1027
max_space_dimension, 1030
OK, 1030
operator<<, 1036
operator∗, 1033
operator∗=, 1033
operator+, 1033, 1034
operator+=, 1034, 1035
operator-, 1035, 1036
operator-=, 1036
Parma_Polyhedra_Library::Congruence, 1037
Parma_Polyhedra_Library::Congruence_System, 1037
Parma_Polyhedra_Library::Constraint, 1037
Parma_Polyhedra_Library::Constraint_System, 1037
Parma_Polyhedra_Library::Generator, 1037
Parma_Polyhedra_Library::Generator_System, 1037
Parma_Polyhedra_Library::Grid, 1037
Parma_Polyhedra_Library::Grid_Generator, 1037
Parma_Polyhedra_Library::Grid_Generator_System, 1038
Parma_Polyhedra_Library::IO_Operators::operator<<, 1038
Parma_Polyhedra_Library::Polyhedron, 1038
Parma_Polyhedra_Library::Scalar_Products, 1038
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Linear_Row, 1055
print, 1030
space_dimension, 1031
sub_mul_assign, 1038
swap, 1032, 1038
total_memory_in_bytes, 1032
zero, 1032
zero_p, 1039
Parma_Polyhedra_Library::Linear_Expression::Linear_Expression
Parma_Polyhedra_Library::Constraint, 579
Parma_Polyhedra_Library::Generator, 716
Parma_Polyhedra_Library::Linear_Row, 1039
~Linear_Row, 1046
all_homogeneous_terms_are_zero, 1046
ascii_dump, 1046
ascii_load, 1047
check_strong_normalized, 1047
coefficient, 1047
compare, 1053
construct, 1047, 1048
flags, 1048
inhomogeneous_term, 1048
is_line_or_equality, 1049
is_necessarily_closed, 1049
is_not_necessarily_closed, 1049
is_ray_or_point_or_inequality, 1049
is_zero, 1050
iter_swap, 1054
Kind, 1044
linear_combine, 1050
Linear_Row, 1045
max_space_dimension, 1050
OK, 1051
operator==, 1055
Parma_Polyhedra_Library::Constraint, 1055
Parma_Polyhedra_Library::Generator, 1055
Parma_Polyhedra_Library::Linear_Expression, 1055
Parma_Polyhedra_Library::Linear_Row::Flags, 667
print, 1051
set_is_line_or_equality, 1051
set_is_ray_or_point_or_inequality, 1052
set_necessarily_closed, 1052
set_not_necessarily_closed, 1052
sign_normalize, 1052
space_dimension, 1052
strong_normalize, 1053
swap, 1055
topology, 1053
Parma_Polyhedra_Library::Linear_Row::Flags, 661
ascii_dump, 664
ascii_load, 664
first_free_bit, 667
Flags, 663, 664
is_line_or_equality, 664
is_necessarily_closed, 665
is_not_necessarily_closed, 665
is_ray_or_point_or_inequality, 665
nnc_bit, 667
nnc_validity_bit, 668
operator==, 666
Parma_Polyhedra_Library::Linear_Row, 667
print, 666
rpi_bit, 668

rpi_validity_bit, 668
set_is_line_or_equality, 666
set_is_ray_or_point_or_inequality, 666
set_necessarily_closed, 666
set_not_necessarily_closed, 667
topology, 667
Parma_Polyhedra_Library::Linear_System, 1056
add_pending_row, 1061, 1062
add_pending_rows, 1062
add_row, 1062
add_rows, 1063
add_rows_and_columns, 1063
ascii_dump, 1063, 1064
ascii_load, 1064
assign_with_pending, 1064
back_substitute, 1064
check_sorted, 1065
clear, 1065
external_memory_in_bytes, 1065
first_pending_row, 1065
gauss, 1066
index_first_pending, 1078
insert, 1066
insert_pending, 1066
is_necessarily_closed, 1067
is_sorted, 1067
Linear_System, 1061
max_space_dimension, 1068
merge_rows_assign, 1068
normalize, 1068
num_lines_or_equalities, 1069
num_pending_rows, 1069
OK, 1069
operator=, 1070
operator==, 1077
permute_columns, 1070
print, 1071
remove_trailing_columns, 1071
resize_no_copy, 1071
row_topology, 1078
set_index_first_pending_row, 1072
set_necessarily_closed, 1072
set_not_necessarily_closed, 1072
set_rows_topology, 1073
set_sorted, 1073
sign_normalize, 1073
simplify, 1074
sort_and_remove_with_sat, 1074
sort_pending_and_remove_duplicates, 1074
sort_rows, 1074, 1075
sorted, 1078
space_dimension, 1075
strong_normalize, 1075
swap, 1076, 1077
topology, 1076
total_memory_in_bytes, 1076
unset_pending_rows, 1076
Parma_Polyhedra_Library::Linear_System::Row_-
Less_Than, 1571
operator(), 1571
Parma_Polyhedra_Library::Linear_System::sort_-
and_remove_with_sat
Parma_Polyhedra_Library::Bit_Matrix, 353
Parma_Polyhedra_Library::Linear_System::With_-
Pending, 1680
Parma_Polyhedra_Library::Matrix, 1078
∼Matrix, 1082
add_recycled_row, 1083
add_row, 1083
add_zero_columns, 1083
add_zero_rows, 1084
add_zero_rows_and_columns, 1084
ascii_dump, 1085
ascii_load, 1085
begin, 1086
clear, 1086
end, 1086
erase_to_end, 1086
external_memory_in_bytes, 1087
has_no_rows, 1087
Matrix, 1082
max_num_columns, 1088
max_num_rows, 1088
num_columns, 1088
num_rows, 1090
OK, 1092
operator=, 1092
operator==, 1095
permute_columns, 1093
print, 1093
remove_trailing_columns, 1093
resize_no_copy, 1094
row_capacity, 1096
row_size, 1096
rows, 1096
swap, 1094, 1095
swap_columns, 1094
total_memory_in_bytes, 1095
Parma_Polyhedra_Library::Matrix::const_iterator,
525
const_iterator, 527
difference_type, 526
i, 529
Iter, 526
iterator_category, 526
operator∗, 527
operator++, 528
operator->, 528

operator=, 528
operator==, 528
pointer, 526
reference, 526
value_type, 527
Parma_Polyhedra_Library::maybe_assign_struct, 1097
function, 1097
Parma_Polyhedra_Library::maybe_assign_struct< Type, Type >, 1097
function, 1098
Parma_Polyhedra_Library::Minus_Infinity, 1098
vclass, 1098
Parma_Polyhedra_Library::MIP_Problem, 1098
~MIP_Problem, 1109
add_constraint, 1109
add_constraints, 1109
add_space_dimensions_and_embed, 1110
add_to_integer_space_dimensions, 1110
ascii_dump, 1111
ascii_load, 1111
base, 1128
choose_branching_variable, 1111
clear, 1112
compute_generator, 1112
compute_simplex_using_exact_pricing, 1112
compute_simplex_using_steepest_edge_float, 1113
compute_tableau, 1113
const_iterator, 1105
Constraint_Sequence, 1105
constraints_begin, 1113
constraints_end, 1113
Control_Parameter_Name, 1106
Control_Parameter_Value, 1106
erase_artificials, 1114
evaluate_objective_function, 1114
external_memory_in_bytes, 1115
external_space_dim, 1128
feasible_point, 1115
first_pending_constraint, 1128
get_control_parameter, 1115
get_exiting_base_index, 1115
i_variables, 1128
initialized, 1128
input_cs, 1129
input_obj_function, 1129
integer_space_dimensions, 1116
internal_space_dim, 1129
is_in_base, 1116
is_lp_satisfiable, 1116
is_mip_satisfiable, 1117
is_satisfiable, 1117
is_satisfied, 1117
is_saturated, 1118
last_generator, 1129
linear_combine, 1118
mapping, 1129
max_space_dimension, 1118
merge_split_variable, 1118
MIP_Problem, 1107–1109
objective_function, 1119
OK, 1119
operator<<, 1127
operator=, 1119
opt_mode, 1130
optimal_value, 1120
optimization_mode, 1120
optimizing_point, 1120
parse_constraints, 1121
pivot, 1122
pricing, 1130
print, 1122
process_pending_constraints, 1122
RAII_Temporary_Real_Relaxation, 1127
second_phase, 1123
set_control_parameter, 1123
set_objective_function, 1123
set_optimization_mode, 1124
solve, 1124
solve_mip, 1124
space_dimension, 1125
Status, 1106
status, 1130
steepest_edge_exact_entering_index, 1125
steepest_edge_float_entering_index, 1126
swap, 1126, 1127
tableau, 1130
textbook_entering_index, 1126
total_memory_in_bytes, 1127
working_cost, 1131
Parma_Polyhedra_Library::MIP_Problem::RAII_Temporary_Real_Relaxation, 1539
~RAII_Temporary_Real_Relaxation, 1540
i_vars, 1540
lp, 1540
RAII_Temporary_Real_Relaxation, 1540
Parma_Polyhedra_Library::Native_Checked_From_Wrapper< Checked_Number< T, P > >, 1131
Policy, 1132
raw_value, 1132
Parma_Polyhedra_Library::Native_Checked_From_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >, 1132
Policy, 1133
raw_value, 1133

Parma_Polyhedra_Library::Native_Checked_To_-
Wrapper< Checked_Number< T, P > >, 1133
Policy, 1134
raw_value, 1134
Parma_Polyhedra_Library::Native_Checked_To_-
Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >, 1134
Policy, 1135
raw_value, 1135
Parma_Polyhedra_Library::NNC_Polyhedron, 1135
~NNC_Polyhedron, 1141
NNC_Polyhedron, 1137–1141
operator=, 1142
poly_hull_assign_if_exact, 1142
upper_bound_assign_if_exact, 1142
Parma_Polyhedra_Library::No_Reduction, 1143
~No_Reduction, 1144
No_Reduction, 1144
product_reduce, 1144
Parma_Polyhedra_Library::Not_A_Number, 1144
vclass, 1145
Parma_Polyhedra_Library::Octagonal_Shape, 1146
~Octagonal_Shape, 1164
add_congruence, 1165
add_congruences, 1165
add_constraint, 1165
add_constraints, 1166
add_octagonal_constraint, 1166, 1167
add_recycled_congruences, 1167
add_recycled_constraints, 1167
add_space_dimensions_and_embed, 1168
add_space_dimensions_and_project, 1168
affine_dimension, 1169
affine_image, 1169
affine_preimage, 1170
ascii_dump, 1171
ascii_load, 1171
BHMZ05_widening_assign, 1171
bounded_affine_image, 1172
bounded_affine_preimage, 1173
bounds, 1173
bounds_from_above, 1174
bounds_from_below, 1174
can_recycle_congruence_systems, 1175
can_recycle_constraint_systems, 1175
CC76_extrapolation_assign, 1175, 1176
CC76_narrowing_assign, 1176
coefficient_type, 1159
coefficient_type_base, 1159
coherent_index, 1219
compute_leaders, 1176, 1177
compute_successors, 1177
concatenate_assign, 1177
congruences, 1178
constrains, 1178
constraints, 1178
contains, 1179
contains_integer_point, 1179
deduce_minus_v_pm_u_bounds, 1180
deduce_v_pm_u_bounds, 1180
default_stop_points, 1225
difference_assign, 1181
drop_some_non_integer_points, 1181, 1182
drop_some_non_integer_points_helper, 1182
euclidean_distance_assign, 1220
expand_space_dimension, 1182
external_memory_in_bytes, 1183
extract_octagonal_difference, 1220
fold_space_dimensions, 1183
forget_all_octagonal_constraints, 1184
forget_binary_octagonal_constraints, 1184
frequency, 1184
generalized_affine_image, 1185, 1186
generalized_affine_preimage, 1186, 1187
get_limiting_octagon, 1188
hash_code, 1188
incremental_strong_closure_assign, 1188
integer_upper_bound_assign_if_exact, 1189
intersection_assign, 1189
is_bounded, 1190
is_discrete, 1190
is_disjoint_from, 1190
is_empty, 1191
is_strong_coherent, 1191
is_strongly_reduced, 1191
is_topologically_closed, 1191
is_universe, 1192
l_infinity_distance_assign, 1221, 1222
limited_BHMZ05_extrapolation_assign, 1192
limited_CC76_extrapolation_assign, 1192
map_space_dimensions, 1193
marked_empty, 1194
marked_strongly_closed, 1195
marked_zero_dim_univ, 1195
matrix, 1225
matrix_at, 1195, 1196
max_min, 1196, 1197
max_space_dimension, 1197
maximize, 1198
minimize, 1199
minimized_congruences, 1200
minimized_constraints, 1200
N, 1160
non_redundant_matrix_entries, 1200
Octagonal_Shape, 1160–1164
OK, 1201

operator<<, 1222
operator=, 1201
operator==, 1222
Parma_Polyhedra_Library::Box, 1223
Parma_Polyhedra_Library::euclidean_-
distance_assign, 1223
Parma_Polyhedra_Library::IO_-
Operators::operator<<, 1223
Parma_Polyhedra_Library::l_infinity_-
distance_assign, 1223
Parma_Polyhedra_Library::Octagonal_Shape,
1223
Parma_Polyhedra_Library::rectilinear_-
distance_assign, 1223
Parma_Polyhedra_Library::Octagonal_Shape,
1223
Parma_Polyhedra_Library::Polyhedron, 1511
print, 1202
rectilinear_distance_assign, 1224
refine, 1202
refine_fp_interval_abstract_store, 1202
refine_no_check, 1203
refine_with_congruence, 1204
refine_with_congruences, 1204
refine_with_constraint, 1204
refine_with_constraints, 1205
relation_with, 1205, 1206
remove_higher_space_dimensions, 1207
remove_space_dimensions, 1207
reset_strongly_closed, 1208
set_empty, 1208
set_strongly_closed, 1209
set_zero_dim_univ, 1209
simplify_using_context_assign, 1209
space_dim, 1226
space_dimension, 1210
status, 1227
strictly_contains, 1210
strong_closure_assign, 1211
strong_coherence_assign, 1211
strong_reduction_assign, 1212
swap, 1212, 1224
throw_constraint_incompatible, 1212
throw_dimension_incompatible, 1213, 1214
throw_expression_too_complex, 1214
throw_generic, 1214
tight_closure_assign, 1215
tight_coherence_would_make_empty, 1215
time_elapse_assign, 1216
topological_closure_assign, 1216
total_memory_in_bytes, 1216
unconstrain, 1216, 1217
upper_bound_assign, 1217
upper_bound_assign_if_exact, 1218
widening_assign, 1218
wrap_assign, 1218
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1644
ascii_dump, 1646
ascii_load, 1646
EMPTY, 1648
flags, 1648
flags_t, 1646
OK, 1646
reset, 1646
reset_empty, 1647
reset_strongly_closed, 1647
reset_zero_dim_univ, 1647
set, 1647
set_empty, 1647
set_strongly_closed, 1647
set_zero_dim_univ, 1647
Status, 1646
STRONGLY_CLOSED, 1648
test_all, 1647
test_any, 1648
test_empty, 1648
test_strongly_closed, 1648
test_zero_dim_univ, 1648
ZERO_DIM_UNIV, 1649
Parma_Polyhedra_Library::OR_Matrix, 1227
~OR_Matrix, 1234
ascii_dump, 1234
ascii_load, 1235
clear, 1235
const_element_iterator, 1232
const_row_iterator, 1232
const_row_reference_type, 1233
element_begin, 1235
element_end, 1236
element_iterator, 1233
euclidean_distance_assign, 1242
external_memory_in_bytes, 1236
grow, 1236
l_infinity_distance_assign, 1242
l_m_distance_assign, 1243
max_num_rows, 1237
num_rows, 1237
OK, 1237
operator<<, 1243
operator=, 1238
operator==, 1243
OR_Matrix, 1233, 1234, 1243
print, 1238
rectilinear_distance_assign, 1243
resize_no_copy, 1238
row_begin, 1239
row_end, 1240

row_first_element_index, 1240
row_iterator, 1233
row_reference_type, 1233
row_size, 1241
shrink, 1241
space_dim, 1244
space_dimension, 1241
swap, 1241, 1244
total_memory_in_bytes, 1242
vec, 1244
vec_capacity, 1245
Parma_Polyhedra_Library::OR_Matrix::any_row_-iterator, 242
any_row_iterator, 245, 250
difference_type, 244
e, 250
i, 250
index, 246
iterator_category, 244
operator<, 248
operator<=, 248
operator>, 249
operator>=, 249
operator∗, 246
operator+, 246
operator++, 246
operator+=, 247
operator-, 247
operator->, 248
operator--, 247, 248
operator-=, 248
operator=, 249
operator==, 249
pointer, 244
reference, 244
row_size, 250
value, 251
value_type, 245
Parma_Polyhedra_Library::OR_Matrix::Pseudo_-Row, 1536
∼Pseudo_Row, 1537
any_row_iterator, 1538
first, 1539
operator=, 1538
OR_Matrix, 1538
Pseudo_Row, 1537–1539
Parma_Polyhedra_Library::Partially_Reduced_-Product, 1245
∼Partially_Reduced_Product, 1259
add_congruence, 1260
add_congruences, 1260
add_constraint, 1260
add_constraints, 1261
add_recycled_congruences, 1261
add_recycled_constraints, 1261
add_space_dimensions_and_embed, 1262
add_space_dimensions_and_project, 1262
affine_dimension, 1262
affine_image, 1263
affine_preimage, 1263
ascii_dump, 1264
ascii_load, 1264
bounded_affine_image, 1264
bounded_affine_preimage, 1265
bounds_from_above, 1265
bounds_from_below, 1265
clear_reduced_flag, 1266
concatenate_assign, 1266
congruences, 1266
constrains, 1267
constraints, 1267
contains, 1267
d1, 1287
d2, 1288
difference_assign, 1268
Domain1, 1254
domain1, 1268
Domain2, 1254
domain2, 1268
drop_some_non_integer_points, 1268, 1269
expand_space_dimension, 1269
external_memory_in_bytes, 1270
fold_space_dimensions, 1270
generalized_affine_image, 1270, 1271
generalized_affine_preimage, 1271, 1272
hash_code, 1272
intersection_assign, 1272
is_bounded, 1273
is_discrete, 1273
is_disjoint_from, 1273
is_empty, 1274
is_reduced, 1274
is_topologically_closed, 1274
is_universe, 1274
map_space_dimensions, 1275
max_space_dimension, 1275
maximize, 1275, 1276
minimize, 1276, 1277
minimized_congruences, 1277
minimized_constraints, 1278
OK, 1278
operator<<, 1286
operator=, 1278
operator==, 1286
Parma_Polyhedra_Library::IO_-Operators::operator<<, 1287
Partially_Reduced_Product, 1255–1259
print, 1279

reduce, 1279
reduced, 1289
refine_with_congruence, 1279
refine_with_congruences, 1280
refine_with_constraint, 1280
refine_with_constraints, 1280
relation_with, 1281
remove_higher_space_dimensions, 1282
remove_space_dimensions, 1282
set_reduced_flag, 1282
space_dimension, 1282
strictly_contains, 1283
swap, 1283, 1287
time_elapse_assign, 1283
topological_closure_assign, 1284
total_memory_in_bytes, 1284
unconstrain, 1284
upper_bound_assign, 1285
upper_bound_assign_if_exact, 1285
widening_assign, 1285
Parma_Polyhedra_Library::PIP_Decision_Node, 1289
~PIP_Decision_Node, 1291
as_decision, 1292
ascii_dump, 1292
ascii_load, 1293
check_ownership, 1293
child_node, 1293
clone, 1294
external_memory_in_bytes, 1294
false_child, 1296
OK, 1294
PIP_Decision_Node, 1292
PIP_Problem::ascii_load, 1296
PIP_Solution_Node, 1296
print_tree, 1294
set_owner, 1295
solve, 1295
total_memory_in_bytes, 1295
true_child, 1297
update_tableau, 1296
Parma_Polyhedra_Library::PIP_Problem, 1297
~PIP_Problem, 1310
add_constraint, 1310
add_constraints, 1310
add_space_dimensions_and_embed, 1311
add_to_parameter_space_dimensions, 1311
ascii_dump, 1312
ascii_load, 1312
big_parameter_dimension, 1319
clear, 1312
const_iterator, 1308
Constraint_Sequence, 1308
constraints_begin, 1313
constraints_end, 1313
Control_Parameter_Name, 1308
Control_Parameter_Value, 1308
control_parameters, 1319
control_parameters_copy, 1313
control_parameters_init, 1313
current_solution, 1319
external_memory_in_bytes, 1313
external_space_dim, 1319
first_pending_constraint, 1319
get_big_parameter_dimension, 1314
get_control_parameter, 1314
initial_context, 1320
input_cs, 1320
internal_space_dim, 1320
is_satisfiable, 1314
max_space_dimension, 1314
OK, 1315
operator<<, 1318
operator=, 1315
optimizing_solution, 1315
parameter_space_dimensions, 1315
parameters, 1320
PIP_Problem, 1309, 1310
PIP_Solution_Node, 1318
print, 1316
print_solution, 1316
set_big_parameter_dimension, 1316
set_control_parameter, 1316
solution, 1316
solve, 1317
space_dimension, 1317
Status, 1308
status, 1320
swap, 1317, 1318
total_memory_in_bytes, 1318
Parma_Polyhedra_Library::PIP_Solution_Node, 1321
~PIP_Solution_Node, 1325
as_solution, 1325
ascii_dump, 1325
ascii_load, 1326
basis, 1331
big_dimension, 1331
check_ownership, 1326
clone, 1326
external_memory_in_bytes, 1326
generate_cut, 1327
mapping, 1331
OK, 1327
parametric_values, 1327
PIP_Problem::ascii_load, 1331
PIP_Solution_Node, 1325
print_tree, 1328

Row_Sign, 1324
row_sign, 1328
set_owner, 1328
sign, 1332
solution, 1332
solution_valid, 1332
solve, 1329
special_equality_row, 1332
tableau, 1333
total_memory_in_bytes, 1329
update_solution, 1330
update_tableau, 1330
var_column, 1333
var_row, 1333
Parma_Polyhedra_Library::PIP_Solution_-Node::No_Constraints, 1143
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1653
~Tableau, 1655
ascii_dump, 1655
ascii_load, 1655
denom, 1658
denominator, 1655
external_memory_in_bytes, 1656
is_better_pivot, 1656
is_integer, 1657
normalize, 1657
OK, 1657
s, 1658
scale, 1657
t, 1658
Tableau, 1655
Parma_Polyhedra_Library::PIP_Tree_Node, 1333
~PIP_Tree_Node, 1338
add_constraint, 1338
art_parameter_begin, 1338
art_parameter_count, 1338
art_parameter_end, 1339
Artificial_Parameter_Sequence, 1337
artificial_parameters, 1346
as_decision, 1339
as_solution, 1339
ascii_dump, 1339
ascii_load, 1340
check_ownership, 1340
clone, 1340
compatibility_check, 1340
Constraint_Sequence, 1337
constraints, 1341
constraints_, 1346
external_memory_in_bytes, 1341
get_owner, 1341
indent_and_print, 1342
OK, 1342
operator<<, 1345
owner_, 1346
parent, 1342
parent_, 1346
parent_merge, 1342
PIP_Decision_Node, 1345
PIP_Problem, 1345
PIP_Solution_Node, 1346
PIP_Tree_Node, 1337, 1338
print, 1343
print_tree, 1343
set_owner, 1344
set_parent, 1344
solve, 1344
total_memory_in_bytes, 1344
update_tableau, 1345
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 251
Artificial_Parameter, 255
ascii_dump, 256
ascii_load, 256
denom, 258
denominator, 256
external_memory_in_bytes, 256
OK, 257
operator<<, 258
operator==, 257
print, 257
swap, 257
total_memory_in_bytes, 258
Parma_Polyhedra_Library::Plus_Infinity, 1347
vclass, 1347
Parma_Polyhedra_Library::Pointset_Powerset, 1347
add_congruence, 1366
add_congruences, 1366
add_constraint, 1366
add_constraints, 1366
add_disjunct, 1367
add_space_dimensions_and_embed, 1367
add_space_dimensions_and_project, 1367
affine_dimension, 1368
affine_image, 1368
affine_preimage, 1368
approximate_partition, 1395
approximate_partition_aux, 1396
ascii_dump, 1369
ascii_load, 1369
Base, 1357
BGP99_extrapolation_assign, 1370
BGP99_heuristics_assign, 1370
BHZ03_widening_assign, 1371
bounded_affine_image, 1371
bounded_affine_preimage, 1372

bounds_from_above, 1372
bounds_from_below, 1373
check_containment, 1396, 1397
collect_certificates, 1373
concatenate_assign, 1373
const_iterator, 1357
const_reverse_iterator, 1357
constrains, 1374
contains, 1374
contains_integer_point, 1374
Det_PSET, 1358
difference_assign, 1375
drop_some_non_integer_points, 1376
element_type, 1358
expand_space_dimension, 1376
external_memory_in_bytes, 1377
fold_space_dimensions, 1377
generalized_affine_image, 1378
generalized_affine_preimage, 1379
geometrically_covers, 1380
geometrically_equals, 1381
hash_code, 1381
intersection_assign, 1382
intersection_preserving_enlarge_element, 1382
is_bounded, 1382
is_cert_multiset_stabilizing, 1382
is_discrete, 1383
is_disjoint_from, 1383
is_empty, 1383
is_topologically_closed, 1383
is_universe, 1384
iterator, 1358
linear_partition, 1397
linear_partition_aux, 1398
map_space_dimensions, 1384
max_space_dimension, 1384
maximize, 1384, 1385
minimize, 1386
OK, 1387
operator=, 1387, 1388
pairwise_reduce, 1388
Pointset_Powerset, 1359–1365
Pointset_Powerset< NNC_Polyhedron >, 1398
print, 1388
refine_with_congruence, 1388
refine_with_congruences, 1389
refine_with_constraint, 1389
refine_with_constraints, 1389
relation_with, 1390
remove_higher_space_dimensions, 1391
remove_space_dimensions, 1391
reverse_iterator, 1358
Sequence, 1358
Sequence_const_iterator, 1359
Sequence_iterator, 1359
simplify_using_context_assign, 1392
size_type, 1359
space_dim, 1399
space_dimension, 1392
strictly_contains, 1392
swap, 1393, 1398
time_elapse_assign, 1393
topological_closure_assign, 1393
total_memory_in_bytes, 1393
unconstrain, 1394
value_type, 1359
widen_fun_ref, 1398
wrap_assign, 1394
Parma_Polyhedra_Library::Poly_Con_Relation, 1399
ascii_dump, 1402
EVERYTHING, 1406
flags, 1406
flags_t, 1402
get_flags, 1402
implies, 1403
IS_DISJOINT, 1406
is_disjoint, 1403
IS_INCLUDED, 1407
is_included, 1403
NOTHING, 1407
nothing, 1404
OK, 1404
operator<<, 1406
operator-, 1405
operator==, 1406
operator&&, 1405
Parma_Polyhedra_Library::IO_-Operators::operator<<, 1406
Poly_Con_Relation, 1402
print, 1404
SATURATES, 1407
saturates, 1404
STRICTLY_INTERSECTS, 1407
strictly_intersects, 1405
Parma_Polyhedra_Library::Poly_Gen_Relation, 1407
ascii_dump, 1410
EVERYTHING, 1413
flags, 1413
flags_t, 1410
get_flags, 1410
implies, 1410, 1411
NOTHING, 1413
nothing, 1411
OK, 1411

operator<<, 1412
operator-, 1412
operator==, 1412
operator&&, 1412
Parma_Polyhedra_Library::IO_-Operators::operator<<, 1413
Poly_Gen_Relation, 1410
print, 1411
SUBSUMES, 1413
subsumes, 1411
Parma_Polyhedra_Library::Polyhedron, 1414
~Polyhedron, 1435
add_and_minimize, 1435, 1436
add_congruence, 1437
add_congruences, 1437
add_constraint, 1438
add_constraints, 1438
add_generator, 1439
add_generators, 1439
add_recycled_congruences, 1440
add_recycled_constraints, 1440
add_recycled_generators, 1441
add_space_dimensions, 1442
add_space_dimensions_and_embed, 1442
add_space_dimensions_and_project, 1443
affine_dimension, 1444
affine_image, 1444
affine_preimage, 1445
ascii_dump, 1447
ascii_load, 1447
BFT00_poly_hull_assign_if_exact, 1447
BHRZ03_combining_constraints, 1448
BHRZ03_evolving_points, 1448
BHRZ03_evolving_rays, 1448
BHRZ03_widening_assign, 1449
BHZ09_C_poly_hull_assign_if_exact, 1449
BHZ09_NNC_poly_hull_assign_if_exact, 1449
BHZ09_poly_hull_assign_if_exact, 1450
bounded_affine_image, 1450
bounded_affine_preimage, 1451
bounded_BHRZ03_extrapolation_assign, 1451
bounded_H79_extrapolation_assign, 1452
bounds, 1452
bounds_from_above, 1453
bounds_from_below, 1453
can_have_something_pending, 1453
can_recycle_congruence_systems, 1453
can_recycle_constraint_systems, 1454
clear_constraints_minimized, 1454
clear_constraints_up_to_date, 1454
clear_empty, 1454
clear_generators_minimized, 1455
clear_generators_up_to_date, 1455
clear_pending_constraints, 1455
clear_pending_generators, 1455
clear_sat_c_up_to_date, 1456
clear_sat_g_up_to_date, 1456
coefficient_type, 1431
con_sys, 1511
concatenate_assign, 1456
congruences, 1457
constrains, 1457
constraints, 1457
constraints_are_minimized, 1458
constraints_are_up_to_date, 1458
contains, 1458
contains_integer_point, 1459
conversion, 1459
difference_assign, 1462
drop_some_non_integer_points, 1463, 1464
expand_space_dimension, 1464
external_memory_in_bytes, 1464
finalize, 1465
fold_space_dimensions, 1465
frequency, 1465
gen_sys, 1512
generalized_affine_image, 1466
generalized_affine_preimage, 1467, 1468
generators, 1468
generators_are_minimized, 1469
generators_are_up_to_date, 1469
grid_generators, 1469
H79_widening_assign, 1470
has_pending_constraints, 1470
has_pending_generators, 1471
has_something_pending, 1471
hash_code, 1471
initialize, 1471
intersection_assign, 1472
is_bounded, 1472
is_discrete, 1472
is_disjoint_from, 1472
is_empty, 1473
is_included_in, 1473
is_necessarily_closed, 1473
is_topologically_closed, 1474
is_universe, 1474
limited_BHRZ03_extrapolation_assign, 1475
limited_H79_extrapolation_assign, 1475
map_space_dimensions, 1476
marked_empty, 1477
max_min, 1477
max_space_dimension, 1478
maximize, 1478, 1479
minimize, 1479–1481
minimized_congruences, 1481

minimized_constraints, 1482
minimized_generators, 1482
minimized_grid_generators, 1482
obtain_sorted_constraints, 1482
obtain_sorted_constraints_with_sat_c, 1483
obtain_sorted_generators, 1483
obtain_sorted_generators_with_sat_g, 1483
OK, 1484
operator<<, 1510
operator=, 1485
operator==, 1510
Parma_Polyhedra_Library::BD_Shape, 1510
Parma_Polyhedra_Library::BHRZ03_-
Certificate, 1510
Parma_Polyhedra_Library::Box, 1510
Parma_Polyhedra_Library::Grid, 1510
Parma_Polyhedra_Library::H79_Certificate,
1510
Parma_Polyhedra_Library::Interfaces::is_-
necessarily_closed_for_interfaces, 1510
Parma_Polyhedra_Library::Octagonal_Shape,
1511
Parma_Polyhedra_Library::Constraint, 586
Parma_Polyhedra_Library::Constraint_-
System, 605
Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Generator_-
System, 744
Parma_Polyhedra_Library::Linear_-
Expression, 1038
poly_difference_assign, 1485
poly_hull_assign, 1485
poly_hull_assign_if_exact, 1511
Polyhedron, 1432–1434
print, 1486
process_pending, 1486
process_pending_constraints, 1486
process_pending_generators, 1487
quick_equivalence_test, 1487
refine_no_check, 1488
refine_with_congruence, 1488
refine_with_congruences, 1488
refine_with_constraint, 1489
refine_with_constraints, 1489
relation_with, 1489, 1490
remove_higher_space_dimensions, 1491
remove_pending_to_obtain_constraints, 1491
remove_pending_to_obtain_generators, 1491
remove_space_dimensions, 1492
sat_c, 1512
sat_c_is_up_to_date, 1492
sat_g, 1512
sat_g_is_up_to_date, 1493
select_CH78_constraints, 1493
select_H79_constraints, 1493
set_constraints_minimized, 1494
set_constraints_pending, 1494
set_constraints_up_to_date, 1494
set_empty, 1494
set_generators_minimized, 1495
set_generators_pending, 1495
set_generators_up_to_date, 1495
set_sat_c_up_to_date, 1495
set_sat_g_up_to_date, 1496
set_zero_dim_univ, 1496
simplified_constraints, 1496
simplify, 1496
simplify_num_saturators_p, 1513
simplify_num_saturators_size, 1513
simplify_using_context_assign, 1497
space_dim, 1513
space_dimension, 1498
status, 1514
strictly_contains, 1498
strongly_minimize_constraints, 1499
strongly_minimize_generators, 1499
swap, 1500, 1511
Three_Valued_Boolean, 1431
throw_dimension_incompatible, 1500–1502
throw_invalid_argument, 1502
throw_invalid_generator, 1503
throw_invalid_generators, 1503
throw_runtime_error, 1503
throw_space_dimension_overflow, 1503
throw_topology_incompatible, 1503, 1504
time_elapse_assign, 1504
topological_closure_assign, 1505
topology, 1505
total_memory_in_bytes, 1506
unconstrain, 1506
update_constraints, 1507
update_generators, 1507
update_sat_c, 1507
update_sat_g, 1508
upper_bound_assign, 1508
widening_assign, 1508
wrap_assign, 1509
Parma_Polyhedra_Library::Polyhedron::Status,
1630
ascii_dump, 1634
ascii_load, 1634
C_MINIMIZED, 1642
C_UP_TO_DATE, 1642
CS_PENDING, 1642
EMPTY, 1642
flags, 1642
flags_t, 1634
G_MINIMIZED, 1643

G_UP_TO_DATE, 1643
get_field, 1642
GS_PENDING, 1643
OK, 1635
print, 1635
reset, 1635
reset_c_minimized, 1635
reset_c_pending, 1635
reset_c_up_to_date, 1636
reset_empty, 1636
reset_g_minimized, 1636
reset_g_pending, 1636
reset_g_up_to_date, 1636
reset_sat_c_up_to_date, 1637
reset_sat_g_up_to_date, 1637
reset_zero_dim_univ, 1637
SAT_C_UP_TO_DATE, 1643
SAT_G_UP_TO_DATE, 1643
set, 1637
set_c_minimized, 1637
set_c_pending, 1637
set_c_up_to_date, 1638
set_empty, 1638
set_g_minimized, 1638
set_g_pending, 1638
set_g_up_to_date, 1638
set_sat_c_up_to_date, 1639
set_sat_g_up_to_date, 1639
set_zero_dim_univ, 1639
Status, 1634
test_all, 1639
test_any, 1639
test_c_minimized, 1640
test_c_pending, 1640
test_c_up_to_date, 1640
test_empty, 1640
test_g_minimized, 1640
test_g_pending, 1641
test_g_up_to_date, 1641
test_sat_c_up_to_date, 1641
test_sat_g_up_to_date, 1641
test_zero_dim_univ, 1641
ZERO_DIM_UNIV, 1643
Parma_Polyhedra_Library::Powerset, 1514
~Powerset, 1522
add_disjunct, 1522
add_non_bottom_disjunct_preserve_-reduction, 1522, 1523
begin, 1523
check_omega_reduced, 1524
clear, 1524
collapse, 1524, 1525
const_iterator, 1520
const_reverse_iterator, 1520
definitely_entails, 1525
drop_disjunct, 1525
drop_disjuncts, 1525
empty, 1526
end, 1526
external_memory_in_bytes, 1527
is_bottom, 1527
is_omega_reduced, 1527
is_top, 1527
iterator, 1520
least_upper_bound_assign, 1528
meet_assign, 1528
OK, 1528
omega_reduce, 1528
operator<<, 1532
operator=, 1529
operator==, 1532
pairwise_apply_assign, 1529
Powerset, 1521, 1522
rbegin, 1530
reduced, 1533
rend, 1530
reverse_iterator, 1520
Sequence, 1520
sequence, 1533
Sequence_const_iterator, 1521
Sequence_iterator, 1521
size, 1530
size_type, 1521
swap, 1531, 1532
total_memory_in_bytes, 1531
upper_bound_assign, 1531
upper_bound_assign_if_exact, 1531
value_type, 1521
Parma_Polyhedra_Library::rectilinear_distance_-assign
Parma_Polyhedra_Library::BD_Shape, 334
Parma_Polyhedra_Library::Octagonal_Shape, 1223
Parma_Polyhedra_Library::Rectilinear_Distance_-Specialization, 1541
combine, 1541
finalize, 1541
Parma_Polyhedra_Library::Recycle_Input, 1541
Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_-Native_Or_Checked< From >::value >::type >, 1547
get, 1547
type, 1547
Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_-Restriction_Integer_Base, typename

From::info_type >::value >::type >, 1548
get, 1548
type, 1548
Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >, 1549
get, 1549
type, 1549
Parma_Polyhedra_Library::Restriction_Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >, 1550
get, 1550
type, 1550
Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_Checked< From >::value >::type >, 1551
get, 1551
type, 1551
Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, typename From::info_type >::value >::type >, 1552
get, 1552
type, 1552
Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_Base, typename From::info_type >::value >::type >, 1553
Parma_Polyhedra_Library::Restriction_Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, typename From::info_type >::value >::type >, 1554
get, 1554
type, 1554
Parma_Polyhedra_Library::Row, 1555
~Row, 1560
allocate, 1560
ascii_dump, 1560
ascii_load, 1561
assign, 1561
construct, 1561
copy_construct_coefficients, 1562
expand_within_capacity, 1562
external_memory_in_bytes, 1562
flags, 1563
iter_swap, 1567
max_size, 1563
normalize, 1563
OK, 1564
operator=, 1564
operator==, 1568
print, 1565
Row, 1558, 1559
shrink, 1565
size, 1566
swap, 1566, 1568
total_memory_in_bytes, 1567
Parma_Polyhedra_Library::Row::Flags, 655
ascii_dump, 658
ascii_load, 658
base_type, 657
bits, 660
first_free_bit, 660
Flags, 658
get_bits, 658
num_bits, 661
operator==, 659
print, 659
reset_bits, 659
Row, 660
set_bits, 660
test_bits, 660
Parma_Polyhedra_Library::Row_Impl_Handler, 1568
~Row_Impl_Handler, 1570
impl, 1570
operator=, 1570
Row_Impl_Handler, 1570
Parma_Polyhedra_Library::Row_Impl_Handler::Impl, 882
~Impl, 885
bump_size, 885
copy_construct_coefficients, 885
expand_within_capacity, 886
external_memory_in_bytes, 886
flags, 886
flags_, 889
Impl, 885
max_size, 886
operator delete, 887
operator new, 887
operator=, 887
set_size, 888
shrink, 888

size, 888
size_, 889
total_memory_in_bytes, 889
vec_, 889
Parma_Polyhedra_Library::Scalar_Products, 1577
assign, 1579, 1580
homogeneous_assign, 1580, 1581
homogeneous_sign, 1582
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Constraint, 586
Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Linear_-Expression, 1038
reduced_assign, 1582, 1583
reduced_sign, 1583, 1584
sign, 1584
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type, 1585
type, 1585
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< char >, 1586
type, 1586
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< float >, 1586
type, 1586
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed char >, 1587
type, 1587
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed int >, 1587
type, 1588
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed long >, 1588
type, 1588
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed short >, 1588
type, 1589
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned char >, 1589
type, 1589
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned int >, 1589
type, 1590
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned long >, 1590
type, 1590
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned long long >, 1591
type, 1591
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned short >, 1591
type, 1591
Parma_Polyhedra_Library::Shape_Preserving_-Reduction, 1592
~Shape_Preserving_Reduction, 1592
product_reduce, 1593
Shape_Preserving_Reduction, 1592
Parma_Polyhedra_Library::Simple_Restriction_-Integer, 1593
get_integer, 1595
integer, 1595
Simple_Restriction_Integer, 1594
Parma_Polyhedra_Library::Simple_Restriction_-Integer_Modulo, 1595
divisor, 1596
modulo_type, 1596
remainder, 1596
Simple_Restriction_Integer_Modulo, 1596
Parma_Polyhedra_Library::Slow_Copy< Checked_Number< T, P > >, 1597
Parma_Polyhedra_Library::Slow_Copy< Interval_-Restriction_Integer_Modulo< T, Base > >, 1598
Parma_Polyhedra_Library::Smash_Reduction, 1599
~Smash_Reduction, 1599
product_reduce, 1600
Smash_Reduction, 1599
Parma_Polyhedra_Library::stdiobuf, 1649
char_type, 1650
fp, 1653
int_type, 1650
overflow, 1651
pbackfail, 1651
stdiobuf, 1651
sync, 1651
traits_type, 1650
uflow, 1651
underflow, 1652
ungetc_buf, 1653
xsgetn, 1652
xsputn, 1652
Parma_Polyhedra_Library::TConstant, 1658
value, 1659
Parma_Polyhedra_Library::Throwable, 1659
~Throwable, 1660
maybe_abandon, 1660
throw_me, 1660
Parma_Polyhedra_Library::Topology_Adjusted_-Scalar_Product_Assign

Parma_Polyhedra_Library::Generator, 723
Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign, 1661
operator(), 1662
Parma_Polyhedra_Library::Constraint, 586
Parma_Polyhedra_Library::Generator, 724
sps_fp, 1663
SPS_type, 1661
Topology_Adjusted_Scalar_Product_Sign, 1662
Parma_Polyhedra_Library::True, 1663
Parma_Polyhedra_Library::Variable, 1664
current_output_function, 1671
default_output_function, 1667
external_memory_in_bytes, 1667
get_output_function, 1667
id, 1667
Init, 1670
less, 1670
max_space_dimension, 1668
OK, 1668
operator<<, 1670
output_function_type, 1666
Parma_Polyhedra_Library::IO_Operators::operator<<, 1670
set_output_function, 1669
space_dimension, 1669
total_memory_in_bytes, 1670
Variable, 1667
varid, 1671
Parma_Polyhedra_Library::Variable::Compare, 483
operator(), 483
Parma_Polyhedra_Library::Variables_Set, 1671
ascii_dump, 1673
ascii_load, 1674
Base, 1673
external_memory_in_bytes, 1674
insert, 1674
max_space_dimension, 1674
OK, 1674
operator<<, 1676
print, 1675
space_dimension, 1675
total_memory_in_bytes, 1675
Variables_Set, 1673
Parma_Polyhedra_Library::Weightwatch_Traits, 1676
check_function, 1678
Delta, 1677
from_delta, 1677
get, 1677
less_than, 1677
Threshold, 1677
weight, 1678
Parma_Polyhedra_Library::Widening_Function, 1678
operator(), 1679
w_method, 1680
Widening_Function, 1679
Widening_Method, 1679
parse_constraints
Parma_Polyhedra_Library::MIP_Problem, 1121
parse_number
Parma_Polyhedra_Library::Checked, 203
parse_number_part
Parma_Polyhedra_Library::Checked, 203
PARTIALLY_SATISFIABLE
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::PIP_Problem, 1309
Partially_Reduced_Product
Parma_Polyhedra_Library::Partially_Reduced_Product, 1255–1259
Partially_Reduced_Product.defs.hh, 1868
Partially_Reduced_Product.inlines.hh, 1870
Partially_Reduced_Product.templates.hh, 1871
Partially_Reduced_Product.types.hh, 1872
pbackfail
Parma_Polyhedra_Library::stdiobuf, 1651
permute_columns
Parma_Polyhedra_Library::Grid_Generator_System, 871
Parma_Polyhedra_Library::Linear_System, 1070
Parma_Polyhedra_Library::Matrix, 1093
Ph_Status.cc, 1873
Ph_Status.idefs.hh, 1873
Ph_Status.inlines.hh, 1873
PIP_Decision_Node
Parma_Polyhedra_Library::PIP_Decision_Node, 1292
Parma_Polyhedra_Library::PIP_Tree_Node, 1345
PIP_Problem
Parma_Polyhedra_Library::PIP_Problem, 1309, 1310
Parma_Polyhedra_Library::PIP_Tree_Node, 1345
PIP_Problem.cc, 1873
PIP_Problem.defs.hh, 1874
PIP_Problem.inlines.hh, 1876
PIP_Problem.templates.hh, 1876
PIP_Problem.types.hh, 1877
PIP_Problem::ascii_load
Parma_Polyhedra_Library::PIP_Decision_Node, 1296

Parma_Polyhedra_Library::PIP_Solution_-Node, 1331
PIP_Problem_Status
PPL_CXX_interface, 103
PIP_Solution_Node
Parma_Polyhedra_Library::PIP_Decision_-Node, 1296
Parma_Polyhedra_Library::PIP_Problem, 1318
Parma_Polyhedra_Library::PIP_Solution_-Node, 1325
Parma_Polyhedra_Library::PIP_Tree_Node, 1346
PIP_Tree
Parma_Polyhedra_Library, 122
PIP_Tree.cc, 1877
PIP_Tree.defs.hh, 1878
PIP_Tree.inlines.hh, 1880
PIP_Tree.types.hh, 1880
PIP_Tree_Node
Parma_Polyhedra_Library::PIP_Tree_Node, 1337, 1338
pivot
Parma_Polyhedra_Library::MIP_Problem, 1122
PIVOT_ROW_STRATEGY
Parma_Polyhedra_Library::PIP_Problem, 1308
PIVOT_ROW_STRATEGY_FIRST
Parma_Polyhedra_Library::PIP_Problem, 1308
PIVOT_ROW_STRATEGY_MAX_COLUMN
Parma_Polyhedra_Library::PIP_Problem, 1308
PLUS_INFINITY
Parma_Polyhedra_Library, 165
plus_infinity
Parma_Polyhedra_Library, 157
Parma_Polyhedra_-Library::Checked::Extended_Int, 650
POINT
Parma_Polyhedra_Library::Generator, 709
Parma_Polyhedra_Library::Grid_Generator, 844
point
Parma_Polyhedra_Library::Generator, 716
pointer
Parma_Polyhedra_Library::const_iterator_-to_const, 556
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 531
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Matrix::const_-iterator, 526
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 244
pointset
Parma_Polyhedra_Library::Determinate, 642
Pointset_Powerset
Parma_Polyhedra_Library::Pointset_-Powerset, 1359–1365
Pointset_Powerset< NNC_Polyhedron >
Parma_Polyhedra_Library::Pointset_-Powerset, 1398
Pointset_Powerset.cc, 1880
Pointset_Powerset.defs.hh, 1881
Pointset_Powerset.inlines.hh, 1882
Pointset_Powerset.templates.hh, 1883
Pointset_Powerset.types.hh, 1885
Policy
Parma_Polyhedra_Library::Native_Checked_-From_Wrapper< Checked_Number< T, P > >, 1132
Parma_Polyhedra_Library::Native_Checked_-From_Wrapper< T, typename Enable_-If< Is_Native< T >::value >::type >, 1133
Parma_Polyhedra_Library::Native_Checked_-To_Wrapper< Checked_Number< T, P > >, 1134
Parma_Polyhedra_Library::Native_Checked_-To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >, 1135
Poly_Con_Relation
Parma_Polyhedra_Library::Poly_Con_-Relation, 1402
Poly_Con_Relation.cc, 1885
Poly_Con_Relation.defs.hh, 1885
Poly_Con_Relation.inlines.hh, 1887
Poly_Con_Relation.types.hh, 1887
poly_difference_assign
Parma_Polyhedra_Library::Polyhedron, 1485
Poly_Gen_Relation
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1410
Poly_Gen_Relation.cc, 1887
Poly_Gen_Relation.defs.hh, 1888
Poly_Gen_Relation.inlines.hh, 1889
Poly_Gen_Relation.types.hh, 1889
poly_hull_assign
Parma_Polyhedra_Library::Polyhedron, 1485
poly_hull_assign_if_exact
Parma_Polyhedra_Library::C_Polyhedron, 443
Parma_Polyhedra_Library::NNC_Polyhedron, 1142

Parma_Polyhedra_Library::Polyhedron, 1511
Polyhedron
Parma_Polyhedra_Library::Polyhedron, 1432–1434
Polyhedron.defs.hh, 1890
PPL_IN_Polyhedron_CLASS, 1892
Polyhedron.inlines.hh, 1892
Polyhedron.templates.hh, 1893
Polyhedron.types.hh, 1894
Polyhedron_chdims.cc, 1894
BE_LAZY, 1894
Polyhedron_nonpublic.cc, 1894
BE_LAZY, 1895
Polyhedron_public.cc, 1895
ENSURE_SORTEDNESS, 1896
Polyhedron_widenings.cc, 1896
POLYNOMIAL_COMPLEXITY
PPL_CXX_interface, 102
POS_INF
Parma_Polyhedra_Library::float_ibm_single, 675
Parma_Polyhedra_Library::float_ieee754_-half, 684
Parma_Polyhedra_Library::float_ieee754_-single, 694
POS_ZERO
Parma_Polyhedra_Library::float_ibm_single, 675
Parma_Polyhedra_Library::float_ieee754_-half, 685
Parma_Polyhedra_Library::float_ieee754_-single, 694
POSITIVE
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
Powerset
Parma_Polyhedra_Library::const_iterator_-to_const, 560
Parma_Polyhedra_Library::iterator_to_const, 996
Parma_Polyhedra_Library::Powerset, 1521, 1522
Powerset.defs.hh, 1898
Powerset.inlines.hh, 1900
Powerset.templates.hh, 1900
Powerset.types.hh, 1902
PPL_CXX_interface
ANY_COMPLEXITY, 102
BITS_128, 102
BITS_16, 102
BITS_32, 102
BITS_64, 102
BITS_8, 102
EMPTY, 102
EQUAL, 103
GREATER_OR_EQUAL, 103
GREATER_THAN, 103
LESS_OR_EQUAL, 103
LESS_THAN, 103
MAXIMIZATION, 103
MINIMIZATION, 103
NECESSARILY_CLOSED, 104
NOT_EQUAL, 103
NOT_NECESSARILY_CLOSED, 104
OPTIMIZED_MIP_PROBLEM, 103
OPTIMIZED_PIP_PROBLEM, 103
OVERFLOW_IMPOSSIBLE, 101
OVERFLOW_UNDEFINED, 101
OVERFLOW_WRAPS, 101
POLYNOMIAL_COMPLEXITY, 102
SIGNED_2_COMPLEMENT, 101
SIMPLEX_COMPLEXITY, 102
UNBOUNDED_MIP_PROBLEM, 102
UNFEASIBLE_MIP_PROBLEM, 102
UNFEASIBLE_PIP_PROBLEM, 103
UNIVERSE, 102
UNSIGNED, 101
PPL_ASSIGN2_SIGNED_SIGNED
checked_int.inlines.hh, 1732
PPL_ASSIGN2_SIGNED_UNSIGNED
checked_int.inlines.hh, 1732
PPL_ASSIGN2_UNSIGNED_SIGNED
checked_int.inlines.hh, 1732
PPL_ASSIGN2_UNSIGNED_UNSIGNED
checked_int.inlines.hh, 1732
PPL_ASSIGN_SIGNED
checked_int.inlines.hh, 1732
PPL_ASSIGN_UNSIGNED
checked_int.inlines.hh, 1732
PPL_BITS_PER_GMP_LIMB
Bit_Row.inlines.hh, 1694
PPL_COMPILE_TIME_CHECK
Parma_Polyhedra_Library::Interval, 922
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 948
Parma_Polyhedra_Library::Is_Same_Or_-Derived, 983
PPL_CXX_interface, 99
PPL_COMPILE_TIME_CHECK_AUX
meta_programming.hh, 1848
PPL_COMPILE_TIME_CHECK_NAME
meta_programming.hh, 1848
PPL_COND
Checked_Number.inlines.hh, 1746
PPL_COND_
Checked_Number.inlines.hh, 1746
PPL_COND_0
Checked_Number.inlines.hh, 1746

PPL_COND_1
Checked_Number.inlines.hh, 1746
PPL_CXX_interface
abandon_expensive_computations, 104
Bounded_Integer_Type_Overflow, 101
Bounded_Integer_Type_Representation, 101
Bounded_Integer_Type_Width, 101
Coefficient, 100
Coefficient_traits, 100
Complexity_Class, 102
const_bool_nodef, 98
const_int_nodef, 98
const_ref_nodef, 98
const_value_nodef, 98
Degenerate_Element, 102
dimension_type, 100
memory_size_type, 101
MIP_Problem_Status, 102
Optimization_Mode, 103
PIP_Problem_Status, 103
PPL_COMPILE_TIME_CHECK, 99
PPL_VERSION, 99
PPL_VERSION_BETA, 99
PPL_VERSION_MAJOR, 99
PPL_VERSION_MINOR, 99
PPL_VERSION_REVISION, 100
Relation_Symbol, 103
Topology, 103
PPL_DB_ROW_EXTRA_DEBUG
DB_Row.defs.hh, 1770
PPL_DECLARE_FUN1_0_0
checked.defs.hh, 1710
Parma_Polyhedra_Library::Checked, 203
PPL_DECLARE_FUN1_0_1
checked.defs.hh, 1710
PPL_DECLARE_FUN1_0_2
checked.defs.hh, 1710
Parma_Polyhedra_Library::Checked, 203
PPL_DECLARE_FUN1_0_3
checked.defs.hh, 1710
PPL_DECLARE_FUN1_1_1
checked.defs.hh, 1710
PPL_DECLARE_FUN1_1_2
checked.defs.hh, 1710
Parma_Polyhedra_Library::Checked, 204
PPL_DECLARE_FUN1_2_2
checked.defs.hh, 1710
PPL_DECLARE_FUN2_0_0
checked.defs.hh, 1711
Parma_Polyhedra_Library::Checked, 204
PPL_DECLARE_FUN2_0_1
checked.defs.hh, 1711
Parma_Polyhedra_Library::Checked, 204, 205
PPL_DECLARE_FUN2_0_2
checked.defs.hh, 1711
Parma_Polyhedra_Library::Checked, 205
PPL_DECLARE_FUN3_0_1
checked.defs.hh, 1711
Parma_Polyhedra_Library::Checked, 206, 207
PPL_DECLARE_FUN5_0_1
checked.defs.hh, 1711
PPL_DECLARE_FUNC1_A
Checked_Number.defs.hh, 1743
PPL_DECLARE_FUNC1_B
Checked_Number.defs.hh, 1743
PPL_DECLARE_FUNC2
Checked_Number.defs.hh, 1743
PPL_DECLARE_FUNC4
Checked_Number.defs.hh, 1744
PPL_DEFINE_ASSIGN_2EXP
Checked_Number.inlines.hh, 1746
PPL_DEFINE_ASSIGN_FUN2_1
Checked_Number.inlines.hh, 1746
PPL_DEFINE_ASSIGN_FUN2_2
Checked_Number.inlines.hh, 1746
PPL_DEFINE_ASSIGN_FUN3_3
Checked_Number.inlines.hh, 1747
PPL_DEFINE_ASSIGN_FUN5_5
Checked_Number.inlines.hh, 1747
PPL_DEFINE_BINARY_OP
Checked_Number.inlines.hh, 1747
PPL_DEFINE_BINARY_OP_ASSIGN
Checked_Number.inlines.hh, 1747
PPL_DEFINE_COMPARE
Checked_Number.inlines.hh, 1747
PPL_DEFINE_COMPARE_OP
Checked_Number.inlines.hh, 1747
PPL_DEFINE_CTOR
Checked_Number.inlines.hh, 1747
PPL_DEFINE_FUNC1_A
Checked_Number.inlines.hh, 1747
PPL_DEFINE_FUNC1_B
Checked_Number.inlines.hh, 1748
PPL_DEFINE_FUNC2
Checked_Number.inlines.hh, 1748
PPL_DEFINE_FUNC4
Checked_Number.inlines.hh, 1748
PPL_DEFINE_INCREMENT
Checked_Number.inlines.hh, 1748
PPL_DIRTY_TEMP_COEFFICIENT
globals.defs.hh, 1788
PPL_FUNCTION_CLASS
checked.defs.hh, 1711
Parma_Polyhedra_Library::Checked, 207
PPL_GT_SILENT
checked.inlines.hh, 1723
PPL_IN_BD_Shape_CLASS
BD_Shape.defs.hh, 1683

PPL_IN_Box_CLASS
Box.defs.hh, 1696
PPL_IN_Grid_CLASS
Grid.defs.hh, 1799
PPL_IN_Octagonal_Shape_CLASS
Octagonal_Shape.defs.hh, 1860
PPL_IN_Polyhedron_CLASS
Polyhedron.defs.hh, 1892
PPL_LT_SILENT
checked.inlines.hh, 1723
PPL_MATCH_GMP_GCDEXT
checked.inlines.hh, 1723
PPL_NAN
Float.defs.hh, 1779
PPL_OR_MATRIX_EXTRA_DEBUG
OR_Matrix.defs.hh, 1865
PPL_OUTPUT_2_PARAM_TEMPLATE_-
DEFINITIONS
globals.defs.hh, 1790
PPL_OUTPUT_3_PARAM_TEMPLATE_-
DEFINITIONS
globals.defs.hh, 1790
PPL_OUTPUT_DECLARATIONS
globals.defs.hh, 1790
PPL_OUTPUT_DEFINITIONS
globals.defs.hh, 1790
PPL_OUTPUT_DEFINITIONS_ASCII_ONLY
globals.defs.hh, 1790
PPL_OUTPUT_TEMPLATE_DEFINITIONS
globals.defs.hh, 1790
PPL_OUTPUT_TEMPLATE_DEFINITIONS_-
ASCII_ONLY
globals.defs.hh, 1790
PPL_PROFILE_ADD_WEIGHT
globals.defs.hh, 1791
PPL_ROW_EXTRA_DEBUG
Row.defs.hh, 1904
PPL_SAFE_CONVERSION
checked.inlines.hh, 1723
Parma_Polyhedra_Library::Checked, 208, 209
ppl_set_GMP_memory_allocation_functions
Init.cc, 1815
PPL_SIMPLEX_USE_MIP_HEURISTIC
MIP_Problem.cc, 1850
PPL_SPECIALIZE_ABS
checked.defs.hh, 1711
PPL_SPECIALIZE_ADD
checked.defs.hh, 1711
PPL_SPECIALIZE_ADD_2EXP
checked.defs.hh, 1712
PPL_SPECIALIZE_ADD_MUL
checked.defs.hh, 1712
PPL_SPECIALIZE_ASSIGN
checked.defs.hh, 1712
PPL_SPECIALIZE_ASSIGN_SPECIAL
checked.defs.hh, 1712
PPL_SPECIALIZE_CEIL
checked.defs.hh, 1712
PPL_SPECIALIZE_CLASSIFY
checked.defs.hh, 1712
PPL_SPECIALIZE_CMP
checked.defs.hh, 1713
PPL_SPECIALIZE_CONSTRUCT
checked.defs.hh, 1713
PPL_SPECIALIZE_CONSTRUCT_SPECIAL
checked.defs.hh, 1713
PPL_SPECIALIZE_COPY
checked.defs.hh, 1713
PPL_SPECIALIZE_DIV
checked.defs.hh, 1713
PPL_SPECIALIZE_DIV_2EXP
checked.defs.hh, 1713
PPL_SPECIALIZE_FLOOR
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_0_0
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_0_1
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_0_2
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_0_3
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_1_1
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_1_2
checked.defs.hh, 1714
PPL_SPECIALIZE_FUN1_2_2
checked.defs.hh, 1715
PPL_SPECIALIZE_FUN2_0_0
checked.defs.hh, 1715
PPL_SPECIALIZE_FUN2_0_1
checked.defs.hh, 1715
PPL_SPECIALIZE_FUN2_0_2
checked.defs.hh, 1715
PPL_SPECIALIZE_FUN3_0_1
checked.defs.hh, 1715
PPL_SPECIALIZE_FUN5_0_1
checked.defs.hh, 1715
PPL_SPECIALIZE_GCD
checked.defs.hh, 1715
PPL_SPECIALIZE_GCDEXT
checked.defs.hh, 1716
PPL_SPECIALIZE_IDIV
checked.defs.hh, 1716
PPL_SPECIALIZE_INPUT
checked.defs.hh, 1716
PPL_SPECIALIZE_IS_INT
checked.defs.hh, 1716

PPL_SPECIALIZE_IS_MINF
checked.defs.hh, 1716
PPL_SPECIALIZE_IS_NAN
checked.defs.hh, 1716
PPL_SPECIALIZE_IS_PINF
checked.defs.hh, 1716
PPL_SPECIALIZE_LCM
checked.defs.hh, 1717
PPL_SPECIALIZE_MUL
checked.defs.hh, 1717
PPL_SPECIALIZE_MUL_2EXP
checked.defs.hh, 1717
PPL_SPECIALIZE_NEG
checked.defs.hh, 1717
PPL_SPECIALIZE_OUTPUT
checked.defs.hh, 1717
PPL_SPECIALIZE_REM
checked.defs.hh, 1717
PPL_SPECIALIZE_SGN
checked.defs.hh, 1718
PPL_SPECIALIZE_SMOD_2EXP
checked.defs.hh, 1718
PPL_SPECIALIZE_SQRT
checked.defs.hh, 1718
PPL_SPECIALIZE_SUB
checked.defs.hh, 1718
PPL_SPECIALIZE_SUB_2EXP
checked.defs.hh, 1718
PPL_SPECIALIZE_SUB_MUL
checked.defs.hh, 1718
PPL_SPECIALIZE_TRUNC
checked.defs.hh, 1719
PPL_SPECIALIZE_UMOD_2EXP
checked.defs.hh, 1719
PPL_STR
globals.defs.hh, 1791
PPL_VERSION
PPL_CXX_interface, 99
PPL_VERSION_BETA
PPL_CXX_interface, 99
PPL_VERSION_MAJOR
PPL_CXX_interface, 99
PPL_VERSION_MINOR
PPL_CXX_interface, 99
PPL_VERSION_REVISION
PPL_CXX_interface, 100
PPL_XSTR
globals.defs.hh, 1791
pred_float
Parma_Polyhedra_Library::Checked, 210
prep
Parma_Polyhedra_Library::Determinate, 644
prepare_inexact
Parma_Polyhedra_Library::Checked, 210
prev
Parma_Polyhedra_Library::Bit_Row, 360
PRICING
Parma_Polyhedra_Library::MIP_Problem, 1106
pricing
Parma_Polyhedra_Library::MIP_Problem, 1130
PRICING_STEEPEST_EDGE_EXACT
Parma_Polyhedra_Library::MIP_Problem, 1106
PRICING_STEEPEST_EDGE_FLOAT
Parma_Polyhedra_Library::MIP_Problem, 1106
PRICING_TEXTBOOK
Parma_Polyhedra_Library::MIP_Problem, 1106
print
Parma_Polyhedra_Library::BD_Shape, 314
Parma_Polyhedra_Library::BD_Shape::Status, 1619
Parma_Polyhedra_Library::Bit_Matrix, 350
Parma_Polyhedra_Library::Box, 412
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Congruence, 498
Parma_Polyhedra_Library::Congruence_System, 519
Parma_Polyhedra_Library::Constraint, 579
Parma_Polyhedra_Library::Constraint_System, 602
Parma_Polyhedra_Library::DB_Matrix, 615
Parma_Polyhedra_Library::Generator, 717
Parma_Polyhedra_Library::Generator_System, 741
Parma_Polyhedra_Library::Grid, 810
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_Library::Grid_Generator, 851
Parma_Polyhedra_Library::Grid_Generator_System, 871
Parma_Polyhedra_Library::Linear_Expression, 1030
Parma_Polyhedra_Library::Linear_Row, 1051
Parma_Polyhedra_Library::Linear_Row::Flags, 666
Parma_Polyhedra_Library::Linear_System, 1071
Parma_Polyhedra_Library::Matrix, 1093
Parma_Polyhedra_Library::MIP_Problem, 1122
Parma_Polyhedra_Library::Octagonal_Shape, 1202
Parma_Polyhedra_Library::OR_Matrix, 1238

Parma_Polyhedra_Library::Partially_-Reduced_Product, 1279
Parma_Polyhedra_Library::PIP_Problem, 1316
Parma_Polyhedra_Library::PIP_Tree_Node, 1343
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 257
Parma_Polyhedra_Library::Pointset_-Powerset, 1388
Parma_Polyhedra_Library::Poly_Con_-Relation, 1404
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1411
Parma_Polyhedra_Library::Polyhedron, 1486
Parma_Polyhedra_-Library::Polyhedron::Status, 1635
Parma_Polyhedra_Library::Row, 1565
Parma_Polyhedra_Library::Row::Flags, 659
Parma_Polyhedra_Library::Variables_Set, 1675
Status, 1626
PRINT_DEBUG_INFO
termination.templates.hh, 1917
print_solution
Parma_Polyhedra_Library::PIP_Problem, 1316
print_tree
Parma_Polyhedra_Library::PIP_Decision_-Node, 1294
Parma_Polyhedra_Library::PIP_Solution_-Node, 1328
Parma_Polyhedra_Library::PIP_Tree_Node, 1343
process_pending
Parma_Polyhedra_Library::Polyhedron, 1486
process_pending_constraints
Parma_Polyhedra_Library::MIP_Problem, 1122
Parma_Polyhedra_Library::Polyhedron, 1486
process_pending_generators
Parma_Polyhedra_Library::Polyhedron, 1487
product_reduce
Parma_Polyhedra_Library::Congruences_-Reduction, 524
Parma_Polyhedra_Library::Constraints_-Reduction, 607
Parma_Polyhedra_Library::No_Reduction, 1144
Parma_Polyhedra_Library::Shape_-Preserving_Reduction, 1593
Parma_Polyhedra_Library::Smash_Reduction, 1600
propagate_constraint
Parma_Polyhedra_Library::Box, 412
propagate_constraint_no_check
Parma_Polyhedra_Library::Box, 413
propagate_constraints
Parma_Polyhedra_Library::Box, 414
propagate_constraints_no_check
Parma_Polyhedra_Library::Box, 415
PROPER_CONGRUENCE
Parma_Polyhedra_Library::Grid, 766
Property
Parma_Polyhedra_Library::Interval, 906
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
pset
Parma_Polyhedra_Library::Determinate::Rep, 1546
Pseudo_Row
Parma_Polyhedra_Library::OR_-Matrix::Pseudo_Row, 1537–1539

quick_equivalence_test
Parma_Polyhedra_Library::Grid, 810
Parma_Polyhedra_Library::Polyhedron, 1487

RAII_Temporary_Real_Relaxation
Parma_Polyhedra_Library::MIP_Problem, 1127
Parma_Polyhedra_Library::MIP_-Problem::RAII_Temporary_Real_-Relaxation, 1540
raw_value
Parma_Polyhedra_Library, 157
Parma_Polyhedra_Library::Checked_Number, 465, 473
Parma_Polyhedra_Library::GMP_Integer, 748
Parma_Polyhedra_Library::Native_Checked_-From_Wrapper< Checked_Number< T, P > >, 1132
Parma_Polyhedra_Library::Native_Checked_-From_Wrapper< T, typename Enable_-If< Is_Native< T >::value >::type >, 1133
Parma_Polyhedra_Library::Native_Checked_-To_Wrapper< Checked_Number< T, P > >, 1134
Parma_Polyhedra_Library::Native_Checked_-To_Wrapper< T, typename Enable_If< Is_Native< T >::value >::type >, 1135
RAY
Parma_Polyhedra_Library::Generator, 709
ray
Parma_Polyhedra_Library::Generator, 717
RAY_OR_POINT_OR_INEQUALITY
Parma_Polyhedra_Library::Linear_Row, 1045

rbegin
Parma_Polyhedra_Library::Powerset, 1530
rectilinear_distance_assign
Parma_Polyhedra_Library::BD_Shape, 335
Parma_Polyhedra_Library::Box, 434
Parma_Polyhedra_Library::DB_Matrix, 618
Parma_Polyhedra_Library::Generator, 724
Parma_Polyhedra_Library::Octagonal_Shape, 1224
Parma_Polyhedra_Library::OR_Matrix, 1243
recycling_insert
Parma_Polyhedra_Library::Congruence_System, 519
Parma_Polyhedra_Library::Grid_Generator_System, 871, 872
reduce
Parma_Polyhedra_Library::Partially_Reduced_Product, 1279
reduce_congruence_with_equality
Parma_Polyhedra_Library::Grid, 810
reduce_equality_with_equality
Parma_Polyhedra_Library::Grid, 810
reduce_line_with_line
Parma_Polyhedra_Library::Grid, 811
reduce_parameter_with_line
Parma_Polyhedra_Library::Grid, 811
reduce_pc_with_pc
Parma_Polyhedra_Library::Grid, 811
reduce_reduced
Parma_Polyhedra_Library::Grid, 812
reduced
Parma_Polyhedra_Library::Partially_Reduced_Product, 1289
Parma_Polyhedra_Library::Powerset, 1533
reduced_assign
Parma_Polyhedra_Library::Scalar_Products, 1582, 1583
reduced_sign
Parma_Polyhedra_Library::Scalar_Products, 1583, 1584
redundancy_dbm
Parma_Polyhedra_Library::BD_Shape, 337
reference
Parma_Polyhedra_Library::const_iterator_to_const, 556
Parma_Polyhedra_Library::DB_Matrix::const_iterator, 531
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Matrix::const_iterator, 526
Parma_Polyhedra_Library::OR_Matrix::any_row_iterator, 244
references
Parma_Polyhedra_Library::Determinate::Rep, 1546
refine
Parma_Polyhedra_Library::BD_Shape, 314
Parma_Polyhedra_Library::Octagonal_Shape, 1202
refine_existential
Parma_Polyhedra_Library::Interval, 922
refine_fp_interval_abstract_store
Parma_Polyhedra_Library::Octagonal_Shape, 1202
refine_interval_no_check
Parma_Polyhedra_Library::Box, 415
refine_no_check
Parma_Polyhedra_Library::BD_Shape, 315
Parma_Polyhedra_Library::Box, 416, 417
Parma_Polyhedra_Library::Grid, 812
Parma_Polyhedra_Library::Octagonal_Shape, 1203
Parma_Polyhedra_Library::Polyhedron, 1488
refine_universal
Parma_Polyhedra_Library::Interval, 923
refine_with_congruence
Parma_Polyhedra_Library::BD_Shape, 316
Parma_Polyhedra_Library::Box, 417
Parma_Polyhedra_Library::Grid, 813
Parma_Polyhedra_Library::Octagonal_Shape, 1204
Parma_Polyhedra_Library::Partially_Reduced_Product, 1279
Parma_Polyhedra_Library::Pointset_Powerset, 1388
Parma_Polyhedra_Library::Polyhedron, 1488
refine_with_congruences
Parma_Polyhedra_Library::BD_Shape, 316
Parma_Polyhedra_Library::Box, 418
Parma_Polyhedra_Library::Grid, 813
Parma_Polyhedra_Library::Octagonal_Shape, 1204
Parma_Polyhedra_Library::Partially_Reduced_Product, 1280
Parma_Polyhedra_Library::Pointset_Powerset, 1389
Parma_Polyhedra_Library::Polyhedron, 1488
refine_with_constraint
Parma_Polyhedra_Library::BD_Shape, 317
Parma_Polyhedra_Library::Box, 418
Parma_Polyhedra_Library::Grid, 813
Parma_Polyhedra_Library::Octagonal_Shape, 1204
Parma_Polyhedra_Library::Partially_Reduced_Product, 1280
Parma_Polyhedra_Library::Pointset_Powerset, 1389

Parma_Polyhedra_Library::Polyhedron, 1489
refine_with_constraints
Parma_Polyhedra_Library::BD_Shape, 317
Parma_Polyhedra_Library::Box, 419
Parma_Polyhedra_Library::Grid, 814
Parma_Polyhedra_Library::Octagonal_Shape, 1205
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1280
Parma_Polyhedra_Library::Pointset_-Powerset, 1389
Parma_Polyhedra_Library::Polyhedron, 1489
Relation_Symbol
PPL_CXX_interface, 103
relation_with
Parma_Polyhedra_Library::BD_Shape, 317, 318
Parma_Polyhedra_Library::Box, 419, 420
Parma_Polyhedra_Library::Generator_-System, 741
Parma_Polyhedra_Library::Grid, 814, 815
Parma_Polyhedra_Library::Octagonal_Shape, 1205, 1206
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1281
Parma_Polyhedra_Library::Pointset_-Powerset, 1390
Parma_Polyhedra_Library::Polyhedron, 1489, 1490
rem_assign
Parma_Polyhedra_Library, 158
Parma_Polyhedra_Library::GMP_Integer, 748
rem_float
Parma_Polyhedra_Library::Checked, 210
rem_mpq
Parma_Polyhedra_Library::Checked, 210
rem_mpz
Parma_Polyhedra_Library::Checked, 210
rem_signed_int
Parma_Polyhedra_Library::Checked, 210
rem_unsigned_int
Parma_Polyhedra_Library::Checked, 211
remainder
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 949
Parma_Polyhedra_Library::Simple_-Restriction_Integer_Modulo, 1596
remove_higher_space_dimensions
Parma_Polyhedra_Library::BD_Shape, 319
Parma_Polyhedra_Library::Box, 421
Parma_Polyhedra_Library::Congruence_-System, 519
Parma_Polyhedra_Library::Grid, 815
Parma_Polyhedra_Library::Grid_Generator_-System, 872
Parma_Polyhedra_Library::Octagonal_Shape, 1207
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1282
Parma_Polyhedra_Library::Pointset_-Powerset, 1391
Parma_Polyhedra_Library::Polyhedron, 1491
remove_inf
Parma_Polyhedra_Library::Interval, 924
remove_invalid_lines_and_rays
Parma_Polyhedra_Library::Generator_-System, 741
remove_pending_to_obtain_constraints
Parma_Polyhedra_Library::Polyhedron, 1491
remove_pending_to_obtain_generators
Parma_Polyhedra_Library::Polyhedron, 1491
remove_space_dimensions
Parma_Polyhedra_Library::BD_Shape, 319
Parma_Polyhedra_Library::Box, 421
Parma_Polyhedra_Library::Grid, 816
Parma_Polyhedra_Library::Grid_Generator_-System, 872
Parma_Polyhedra_Library::Octagonal_Shape, 1207
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1282
Parma_Polyhedra_Library::Pointset_-Powerset, 1391
Parma_Polyhedra_Library::Polyhedron, 1492
remove_sup
Parma_Polyhedra_Library::Interval, 924
remove_trailing_columns
Parma_Polyhedra_Library::Linear_System, 1071
Parma_Polyhedra_Library::Matrix, 1093
rend
Parma_Polyhedra_Library::Powerset, 1530
Rep
Parma_Polyhedra_Library::Determinate::Rep, 1543, 1544
reset
Parma_Polyhedra_Library::BD_-Shape::Status, 1619
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1646
Parma_Polyhedra_-Library::Polyhedron::Status, 1635
Status, 1626
reset_bit
Parma_Polyhedra_Library::Interval_NS, 239

reset_bits
Parma_Polyhedra_Library::Interval_NS, 239
Parma_Polyhedra_Library::Row::Flags, 659
reset_c_minimized
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_-Library::Polyhedron::Status, 1635
Status, 1626
reset_c_pending
Parma_Polyhedra_Library::Grid::Status, 1604
Parma_Polyhedra_-Library::Polyhedron::Status, 1635
Status, 1626
reset_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1636
Status, 1626
reset_empty
Parma_Polyhedra_Library::BD_-Shape::Status, 1619
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
Parma_Polyhedra_-Library::Polyhedron::Status, 1636
Status, 1626
reset_empty_up_to_date
Parma_Polyhedra_Library::Box, 421
Parma_Polyhedra_Library::Box::Status, 1613
reset_g_minimized
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1636
Status, 1626
reset_g_pending
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1636
Status, 1626
reset_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1636
Status, 1626
reset_sat_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1637
Status, 1626
reset_sat_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-Library::Polyhedron::Status, 1637
Status, 1627
reset_shortest_path_closed
Parma_Polyhedra_Library::BD_Shape, 320
Parma_Polyhedra_Library::BD_-Shape::Status, 1619
reset_shortest_path_reduced
Parma_Polyhedra_Library::BD_Shape, 320
Parma_Polyhedra_Library::BD_-Shape::Status, 1619
reset_strongly_closed
Parma_Polyhedra_Library::Octagonal_Shape, 1208
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
reset_universe
Parma_Polyhedra_Library::Box::Status, 1613
reset_zero_dim_univ
Parma_Polyhedra_Library::BD_-Shape::Status, 1619
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
Parma_Polyhedra_-Library::Polyhedron::Status, 1637
Status, 1627
resize
Parma_Polyhedra_Library::Bit_Matrix, 351
resize_no_copy
Parma_Polyhedra_Library::Congruence_-System, 519
Parma_Polyhedra_Library::DB_Matrix, 615
Parma_Polyhedra_Library::Grid_Generator_-System, 873
Parma_Polyhedra_Library::Linear_System, 1071
Parma_Polyhedra_Library::Matrix, 1094
Parma_Polyhedra_Library::OR_Matrix, 1238
restore_pre_PPL_rounding
Parma_Polyhedra_Library, 158
Parma_Polyhedra_Library::Init, 899
restrict
Parma_Polyhedra_Library::Interval_-Restriction_Integer, 945
Parma_Polyhedra_Library::Interval_-Restriction_Integer_Modulo, 948
Parma_Polyhedra_Library::Interval_-Restriction_None_Base, 952
Result
Parma_Polyhedra_Library::Checked, 222
Result_Class
Parma_Polyhedra_Library::Checked, 223
Result_Relation

Parma_Polyhedra_Library::Checked, 223
result_relation
Parma_Polyhedra_Library::Checked, 211
reverse_iterator
Parma_Polyhedra_Library::Pointset_-Powerset, 1358
Parma_Polyhedra_Library::Powerset, 1520
rint
Parma_Polyhedra_Library::Checked, 211
round
Parma_Polyhedra_Library::Checked, 211
ROUND_DEFAULT_CONSTRUCTOR
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
ROUND_DEFAULT_FUNCTION
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
ROUND_DEFAULT_INPUT
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
ROUND_DEFAULT_OPERATOR
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
ROUND_DEFAULT_OUTPUT
Parma_Polyhedra_Library::Checked_-Number_Transparent_Policy, 479
round_gt_float
Parma_Polyhedra_Library::Checked, 211
round_gt_int
Parma_Polyhedra_Library::Checked, 212
round_gt_int_no_overflow
Parma_Polyhedra_Library::Checked, 212
round_gt_mpz
Parma_Polyhedra_Library::Checked, 212
round_lt_float
Parma_Polyhedra_Library::Checked, 212
round_lt_int
Parma_Polyhedra_Library::Checked, 212
round_lt_int_no_overflow
Parma_Polyhedra_Library::Checked, 212
round_lt_mpz
Parma_Polyhedra_Library::Checked, 213
rounding_dir
Parma_Polyhedra_Library, 158
Row
Parma_Polyhedra_Library::Row, 1558, 1559
Parma_Polyhedra_Library::Row::Flags, 660
Row.cc, 1902
Row.defs.hh, 1903
PPL_ROW_EXTRA_DEBUG, 1904
Row.inlines.hh, 1904
Row.types.hh, 1905
row_begin
Parma_Polyhedra_Library::OR_Matrix, 1239
row_capacity
Parma_Polyhedra_Library::DB_Matrix, 618
Parma_Polyhedra_Library::Matrix, 1096
row_end
Parma_Polyhedra_Library::OR_Matrix, 1240
row_first_element_index
Parma_Polyhedra_Library::OR_Matrix, 1240
Row_Impl_Handler
Parma_Polyhedra_Library::Row_Impl_-Handler, 1570
row_iterator
Parma_Polyhedra_Library::OR_Matrix, 1233
row_reference_type
Parma_Polyhedra_Library::OR_Matrix, 1233
Row_Sign
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
row_sign
Parma_Polyhedra_Library::PIP_Solution_-Node, 1328
row_size
Parma_Polyhedra_Library::Bit_Matrix, 353
Parma_Polyhedra_Library::DB_Matrix, 619
Parma_Polyhedra_Library::Matrix, 1096
Parma_Polyhedra_Library::OR_Matrix, 1241
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 250
row_topology
Parma_Polyhedra_Library::Linear_System, 1078
rows
Parma_Polyhedra_Library::Bit_Matrix, 354
Parma_Polyhedra_Library::DB_Matrix, 619
Parma_Polyhedra_Library::Matrix, 1096
rows_are_zero
Parma_Polyhedra_Library::Grid, 816
rows_erase_to_end
Parma_Polyhedra_Library::Bit_Matrix, 351
rpi_bit
Parma_Polyhedra_Library::Linear_-Row::Flags, 668
rpi_validity_bit
Parma_Polyhedra_Library::Linear_-Row::Flags, 668

s
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1658
sat_c
Parma_Polyhedra_Library::Polyhedron, 1512
sat_c_is_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1492
SAT_C_UP_TO_DATE
Parma_Polyhedra_Library::Grid::Status, 1609

Parma_Polyhedra_-
Library::Polyhedron::Status, 1643
Status, 1630
sat_g
Parma_Polyhedra_Library::Polyhedron, 1512
sat_g_is_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1493
SAT_G_UP_TO_DATE
Parma_Polyhedra_Library::Grid::Status, 1609
Parma_Polyhedra_-
Library::Polyhedron::Status, 1643
Status, 1630
SATISFIABLE
Parma_Polyhedra_Library::MIP_Problem,
1106
satisfied_by_all_generators
Parma_Polyhedra_Library::Generator_-
System, 741
satisfied_by_all_generators_C
Parma_Polyhedra_Library::Generator_-
System, 742
satisfied_by_all_generators_NNC
Parma_Polyhedra_Library::Generator_-
System, 742
satisfies_all_congruences
Parma_Polyhedra_Library::Congruence_-
System, 520
satisfies_all_constraints
Parma_Polyhedra_Library::Constraint_-
System, 602
SATURATES
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1407
saturates
Parma_Polyhedra_Library::Poly_Con_-
Relation, 1404
Scalar_As_Interval_Info
Parma_Polyhedra_Library::Interval_NS, 236
Scalar_As_Interval_Info_Open
Parma_Polyhedra_Library::Interval_NS, 236
SCALAR_INFO
Parma_Polyhedra_Library::Interval_NS, 240
Scalar_Products
Parma_Polyhedra_Library::Grid_Generator,
856
Scalar_Products.cc, 1905
Scalar_Products.defs.hh, 1905
Scalar_Products.inlines.hh, 1906
Scalar_Products.types.hh, 1907
scale
Parma_Polyhedra_Library::PIP_Solution_-
Node::Tableau, 1657
scale_to_divisor
Parma_Polyhedra_Library::Grid_Generator,
851
second_phase
Parma_Polyhedra_Library::MIP_Problem,
1123
select_CH78_constraints
Parma_Polyhedra_Library::Polyhedron, 1493
select_H79_constraints
Parma_Polyhedra_Library::Polyhedron, 1493
select_wider_congruences
Parma_Polyhedra_Library::Grid, 817
select_wider_generators
Parma_Polyhedra_Library::Grid, 817
seq
Parma_Polyhedra_Library::Box, 435
Sequence
Parma_Polyhedra_Library::Box, 378
Parma_Polyhedra_Library::Pointset_-
Powerset, 1358
Parma_Polyhedra_Library::Powerset, 1520
sequence
Parma_Polyhedra_Library::Powerset, 1533
Sequence_const_iterator
Parma_Polyhedra_Library::Pointset_-
Powerset, 1359
Parma_Polyhedra_Library::Powerset, 1521
Sequence_iterator
Parma_Polyhedra_Library::Pointset_-
Powerset, 1359
Parma_Polyhedra_Library::Powerset, 1521
set
Parma_Polyhedra_Library::BD_-
Shape::Status, 1619
Parma_Polyhedra_Library::Bit_Row, 360
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1647
Parma_Polyhedra_-
Library::Polyhedron::Status, 1637
Status, 1627
set_big_parameter_dimension
Parma_Polyhedra_Library::PIP_Problem,
1316
set_bit
Parma_Polyhedra_Library::Interval_NS, 239
set_bits
Parma_Polyhedra_Library::Interval_NS, 239
Parma_Polyhedra_Library::Row::Flags, 660
set_boundary_property
Parma_Polyhedra_Library::Interval_Info_-
Bitset, 936
Parma_Polyhedra_Library::Interval_Info_-
Null, 940

set_c_minimized
Parma_Polyhedra_Library::Grid::Status, 1605
Parma_Polyhedra_-
Library::Polyhedron::Status, 1637
Status, 1627
set_c_pending
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_-
Library::Polyhedron::Status, 1637
Status, 1627
set_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_-
Library::Polyhedron::Status, 1638
Status, 1627
set_congruences_minimized
Parma_Polyhedra_Library::Grid, 817
set_congruences_up_to_date
Parma_Polyhedra_Library::Grid, 817
set_constraints_minimized
Parma_Polyhedra_Library::Polyhedron, 1494
set_constraints_pending
Parma_Polyhedra_Library::Polyhedron, 1494
set_constraints_up_to_date
Parma_Polyhedra_Library::Polyhedron, 1494
set_control_parameter
Parma_Polyhedra_Library::MIP_Problem,
1123
Parma_Polyhedra_Library::PIP_Problem,
1316
set_difference
Parma_Polyhedra_Library::Bit_Row, 362
set_divisor
Parma_Polyhedra_Library::Grid_Generator,
852
set_empty
Parma_Polyhedra_Library::BD_Shape, 321
Parma_Polyhedra_Library::BD_-
Shape::Status, 1619
Parma_Polyhedra_Library::Box, 422
Parma_Polyhedra_Library::Box::Status, 1613
Parma_Polyhedra_Library::Grid, 818
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_Library::Octagonal_Shape,
1208
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1647
Parma_Polyhedra_Library::Polyhedron, 1494
Parma_Polyhedra_-
Library::Polyhedron::Status, 1638
Status, 1627
set_empty_up_to_date
Parma_Polyhedra_Library::Box, 422
Parma_Polyhedra_Library::Box::Status, 1614
set_g_minimized
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_-
Library::Polyhedron::Status, 1638
Status, 1627
set_g_pending
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_-
Library::Polyhedron::Status, 1638
Status, 1627
set_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_-
Library::Polyhedron::Status, 1638
Status, 1627
set_generators_minimized
Parma_Polyhedra_Library::Grid, 818
Parma_Polyhedra_Library::Polyhedron, 1495
set_generators_pending
Parma_Polyhedra_Library::Polyhedron, 1495
set_generators_up_to_date
Parma_Polyhedra_Library::Grid, 818
Parma_Polyhedra_Library::Polyhedron, 1495
set_index_first_pending_row
Parma_Polyhedra_Library::Grid_Generator_-
System, 873
Parma_Polyhedra_Library::Linear_System,
1072
set_infinities
Parma_Polyhedra_Library::Interval, 924
set_integer
Parma_Polyhedra_Library, 158
Parma_Polyhedra_Library::Interval_-
Restriction_Integer, 945
set_intersection
Parma_Polyhedra_Library::Bit_Row, 362
set_interval
Parma_Polyhedra_Library::Box, 422
set_interval_property
Parma_Polyhedra_Library::Interval_Info_-
Bitset, 936
Parma_Polyhedra_Library::Interval_Info_-
Null, 940
set_irrational_precision
Parma_Polyhedra_Library, 158
set_is_equality
Parma_Polyhedra_Library::Congruence, 498
Parma_Polyhedra_Library::Constraint, 579
set_is_inequality
Parma_Polyhedra_Library::Constraint, 579
set_is_line
Parma_Polyhedra_Library::Generator, 717
Parma_Polyhedra_Library::Grid_Generator,
852

set_is_line_or_equality
Parma_Polyhedra_Library::Linear_Row, 1051
Parma_Polyhedra_Library::Linear_-Row::Flags, 666
set_is_parameter
Parma_Polyhedra_Library::Grid_Generator, 852
set_is_parameter_or_point
Parma_Polyhedra_Library::Grid_Generator, 852
set_is_ray_or_point
Parma_Polyhedra_Library::Generator, 717
set_is_ray_or_point_or_inequality
Parma_Polyhedra_Library::Linear_Row, 1052
Parma_Polyhedra_Library::Linear_-Row::Flags, 666
set_max
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 697
set_mp_size
Parma_Polyhedra_Library::Checked, 213
set_necessarily_closed
Parma_Polyhedra_Library::Linear_Row, 1052
Parma_Polyhedra_Library::Linear_-Row::Flags, 666
Parma_Polyhedra_Library::Linear_System, 1072
set_neg_overflow_float
Parma_Polyhedra_Library::Checked, 213
set_neg_overflow_int
Parma_Polyhedra_Library::Checked, 213
set_nonempty
Parma_Polyhedra_Library::Box, 423
set_not_necessarily_closed
Parma_Polyhedra_Library::Linear_Row, 1052
Parma_Polyhedra_Library::Linear_-Row::Flags, 667
Parma_Polyhedra_Library::Linear_System, 1072
set_objective_function
Parma_Polyhedra_Library::MIP_Problem, 1123
set_optimization_mode
Parma_Polyhedra_Library::MIP_Problem, 1124
set_output_function
Parma_Polyhedra_Library::Variable, 1669
set_owner
Parma_Polyhedra_Library::PIP_Decision_-Node, 1295
Parma_Polyhedra_Library::PIP_Solution_-Node, 1328
Parma_Polyhedra_Library::PIP_Tree_Node, 1344
set_parent
Parma_Polyhedra_Library::PIP_Tree_Node, 1344
set_pos_overflow_float
Parma_Polyhedra_Library::Checked, 213
set_pos_overflow_int
Parma_Polyhedra_Library::Checked, 213
set_reduced_flag
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1282
set_rounding_for_PPL
Parma_Polyhedra_Library, 159
Parma_Polyhedra_Library::Init, 899
set_rows_topology
Parma_Polyhedra_Library::Linear_System, 1073
set_sat_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_Library::Polyhedron, 1495
Parma_Polyhedra_-Library::Polyhedron::Status, 1639
Status, 1627
set_sat_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_Library::Polyhedron, 1496
Parma_Polyhedra_-Library::Polyhedron::Status, 1639
Status, 1628
set_shortest_path_closed
Parma_Polyhedra_Library::BD_Shape, 321
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
set_shortest_path_reduced
Parma_Polyhedra_Library::BD_Shape, 321
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
set_size
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 896
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 888
set_sorted

Parma_Polyhedra_Library::Grid_Generator_-System, 873
Parma_Polyhedra_Library::Linear_System, 1073
set_strongly_closed
Parma_Polyhedra_Library::Octagonal_Shape, 1209
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
set_union
Parma_Polyhedra_Library::Bit_Row, 362
set_universe
Parma_Polyhedra_Library::Box::Status, 1614
set_unrestricted
Parma_Polyhedra_Library, 159
set_until
Parma_Polyhedra_Library::Bit_Row, 360
set_zero_dim_univ
Parma_Polyhedra_Library::BD_Shape, 322
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
Parma_Polyhedra_Library::Grid, 819
Parma_Polyhedra_Library::Grid::Status, 1606
Parma_Polyhedra_Library::Octagonal_Shape, 1209
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
Parma_Polyhedra_Library::Polyhedron, 1496
Parma_Polyhedra_-Library::Polyhedron::Status, 1639
Status, 1628
sgn
Parma_Polyhedra_Library::Checked_Number, 473
sgn_float
Parma_Polyhedra_Library::Checked, 214
sgn_generic
Parma_Polyhedra_Library::Checked, 214
SGN_MASK
Parma_Polyhedra_Library::float_ibm_single, 675
Parma_Polyhedra_Library::float_ieee754_-half, 685
Parma_Polyhedra_Library::float_ieee754_-single, 694
sgn_mp
Parma_Polyhedra_Library::Checked, 214
Shape_Preserving_Product
Parma_Polyhedra_Library::Domain_Product, 645
Shape_Preserving_Reduction
Parma_Polyhedra_Library::Shape_-Preserving_Reduction, 1592
shift_unprimed_variables
Parma_Polyhedra_-Library::Implementation::Termination, 231
SHORTEST_PATH_CLOSED
Parma_Polyhedra_Library::BD_-Shape::Status, 1621
shortest_path_closure_assign
Parma_Polyhedra_Library::BD_Shape, 322
SHORTEST_PATH_REDUCED
Parma_Polyhedra_Library::BD_-Shape::Status, 1621
shortest_path_reduction_assign
Parma_Polyhedra_Library::BD_Shape, 323
shrink
Parma_Polyhedra_Library::DB_Row, 630
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 896
Parma_Polyhedra_Library::OR_Matrix, 1241
Parma_Polyhedra_Library::Row, 1565
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 888
shrink_to_congruence_no_check
Parma_Polyhedra_Library, 159
sign
Parma_Polyhedra_Library::PIP_Solution_-Node, 1332
Parma_Polyhedra_Library::Scalar_Products, 1584
sign_bit
Parma_Polyhedra_Library::float_ibm_single, 673
Parma_Polyhedra_Library::float_ieee754_-double, 678
Parma_Polyhedra_Library::float_ieee754_-half, 683
Parma_Polyhedra_Library::float_ieee754_-quad, 687
Parma_Polyhedra_Library::float_ieee754_-single, 692
Parma_Polyhedra_Library::float_intel_-double_extended, 697
sign_normalize
Parma_Polyhedra_Library::Congruence, 499
Parma_Polyhedra_Library::Linear_Row, 1052
Parma_Polyhedra_Library::Linear_System, 1073
SIGNED_2_COMPLEMENT
PPL_CXX_interface, 101
Simple_Restriction_Integer
Parma_Polyhedra_Library::Simple_-Restriction_Integer, 1594
Simple_Restriction_Integer_Modulo
Parma_Polyhedra_Library::Simple_-Restriction_Integer_Modulo, 1596

SIMPLEX_COMPLEXITY
PPL_CXX_interface, 102
simplified_constraints
Parma_Polyhedra_Library::Polyhedron, 1496
simplify
Parma_Polyhedra_Library::Constraint_-System, 603
Parma_Polyhedra_Library::Generator_-System, 742
Parma_Polyhedra_Library::Grid, 819
Parma_Polyhedra_Library::Linear_System, 1074
Parma_Polyhedra_Library::Polyhedron, 1496
simplify.cc, 1907
simplify_num_saturators_p
Parma_Polyhedra_Library::Polyhedron, 1513
simplify_num_saturators_size
Parma_Polyhedra_Library::Polyhedron, 1513
simplify_using_context_assign
Parma_Polyhedra_Library::BD_Shape, 323
Parma_Polyhedra_Library::Box, 423
Parma_Polyhedra_Library::Grid, 820
Parma_Polyhedra_Library::Interval, 924
Parma_Polyhedra_Library::Octagonal_Shape, 1209
Parma_Polyhedra_Library::Pointset_-Powerset, 1392
Parma_Polyhedra_Library::Polyhedron, 1497
size
Parma_Polyhedra_Library::DB_Row, 630
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 896
Parma_Polyhedra_Library::Grid_Generator, 853
Parma_Polyhedra_Library::Powerset, 1530
Parma_Polyhedra_Library::Row, 1566
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 888
size_
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 897
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 889
size_type
Parma_Polyhedra_Library::Pointset_-Powerset, 1359
Parma_Polyhedra_Library::Powerset, 1521
skip_forward
Parma_Polyhedra_Library::Congruence_-System::const_iterator, 543
Parma_Polyhedra_Library::Constraint_-System::const_iterator, 538
Parma_Polyhedra_Library::Generator_-System::const_iterator, 548
Smash_Product
Parma_Polyhedra_Library::Domain_Product, 645
Smash_Reduction
Parma_Polyhedra_Library::Smash_Reduction, 1599
smod_2exp_float
Parma_Polyhedra_Library::Checked, 214
smod_2exp_mpq
Parma_Polyhedra_Library::Checked, 214
smod_2exp_mpz
Parma_Polyhedra_Library::Checked, 214
smod_2exp_signed_int
Parma_Polyhedra_Library::Checked, 214
smod_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 215
solution
Parma_Polyhedra_Library::PIP_Problem, 1316
Parma_Polyhedra_Library::PIP_Solution_-Node, 1332
solution_valid
Parma_Polyhedra_Library::PIP_Solution_-Node, 1332
solve
Parma_Polyhedra_Library::MIP_Problem, 1124
Parma_Polyhedra_Library::PIP_Decision_-Node, 1295
Parma_Polyhedra_Library::PIP_Problem, 1317
Parma_Polyhedra_Library::PIP_Solution_-Node, 1329
Parma_Polyhedra_Library::PIP_Tree_Node, 1344
solve_mip
Parma_Polyhedra_Library::MIP_Problem, 1124
sort_and_remove_with_sat
Parma_Polyhedra_Library::Linear_System, 1074
sort_pending_and_remove_duplicates
Parma_Polyhedra_Library::Linear_System, 1074
sort_rows
Parma_Polyhedra_Library::Bit_Matrix, 351
Parma_Polyhedra_Library::Linear_System, 1074, 1075
sorted
Parma_Polyhedra_Library::Linear_System, 1078
sorted_contains
Parma_Polyhedra_Library::Bit_Matrix, 351
space_dim

Parma_Polyhedra_Library::Grid, 831
Parma_Polyhedra_Library::Octagonal_Shape, 1226
Parma_Polyhedra_Library::OR_Matrix, 1244
Parma_Polyhedra_Library::Pointset_Powerset, 1399
Parma_Polyhedra_Library::Polyhedron, 1513
space_dimension
Parma_Polyhedra_Library::BD_Shape, 324
Parma_Polyhedra_Library::Box, 423
Parma_Polyhedra_Library::Congruence, 499
Parma_Polyhedra_Library::Congruence_System, 520
Parma_Polyhedra_Library::Constraint, 579
Parma_Polyhedra_Library::Constraint_System, 603
Parma_Polyhedra_Library::Generator, 718
Parma_Polyhedra_Library::Generator_System, 742
Parma_Polyhedra_Library::Grid, 820
Parma_Polyhedra_Library::Grid_Generator, 853
Parma_Polyhedra_Library::Grid_Generator_System, 873
Parma_Polyhedra_Library::Linear_Expression, 1031
Parma_Polyhedra_Library::Linear_Row, 1052
Parma_Polyhedra_Library::Linear_System, 1075
Parma_Polyhedra_Library::MIP_Problem, 1125
Parma_Polyhedra_Library::Octagonal_Shape, 1210
Parma_Polyhedra_Library::OR_Matrix, 1241
Parma_Polyhedra_Library::Partially_Reduced_Product, 1282
Parma_Polyhedra_Library::PIP_Problem, 1317
Parma_Polyhedra_Library::Pointset_Powerset, 1392
Parma_Polyhedra_Library::Polyhedron, 1498
Parma_Polyhedra_Library::Variable, 1669
Parma_Polyhedra_Library::Variables_Set, 1675
special_equality_row
Parma_Polyhedra_Library::PIP_Solution_Node, 1332
sps_fp
Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign, 1663
SPS_type
Parma_Polyhedra_Library::Topology_Adjusted_Scalar_Product_Sign, 1661
sqrt_assign
Parma_Polyhedra_Library, 159
Parma_Polyhedra_Library::Checked_Number, 474
Parma_Polyhedra_Library::GMP_Integer, 749
sqrt_float
Parma_Polyhedra_Library::Checked, 215
sqrt_mpq
Parma_Polyhedra_Library::Checked, 215
sqrt_mpz
Parma_Polyhedra_Library::Checked, 215
sqrt_signed_int
Parma_Polyhedra_Library::Checked, 215
sqrt_unsigned_int
Parma_Polyhedra_Library::Checked, 216
Status, 1622
ascii_dump, 1625
ascii_load, 1625
C_MINIMIZED, 1629
C_UP_TO_DATE, 1629
CS_PENDING, 1629
EMPTY, 1629
flags, 1629
flags_t, 1625
G_MINIMIZED, 1629
G_UP_TO_DATE, 1630
GS_PENDING, 1630
OK, 1625
Parma_Polyhedra_Library::BD_Shape::Status, 1618
Parma_Polyhedra_Library::Box::Status, 1612
Parma_Polyhedra_Library::Grid::Status, 1603
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::Octagonal_Shape::Status, 1646
Parma_Polyhedra_Library::PIP_Problem, 1308
Parma_Polyhedra_Library::Polyhedron::Status, 1634
print, 1626
reset, 1626
reset_c_minimized, 1626
reset_c_pending, 1626
reset_c_up_to_date, 1626
reset_empty, 1626
reset_g_minimized, 1626
reset_g_pending, 1626
reset_g_up_to_date, 1626
reset_sat_c_up_to_date, 1626
reset_sat_g_up_to_date, 1627
reset_zero_dim_univ, 1627
SAT_C_UP_TO_DATE, 1630
SAT_G_UP_TO_DATE, 1630
set, 1627

set_c_minimized, 1627
set_c_pending, 1627
set_c_up_to_date, 1627
set_empty, 1627
set_g_minimized, 1627
set_g_pending, 1627
set_g_up_to_date, 1627
set_sat_c_up_to_date, 1627
set_sat_g_up_to_date, 1628
set_zero_dim_univ, 1628
Status, 1625
test_all, 1628
test_any, 1628
test_c_minimized, 1628
test_c_pending, 1628
test_c_up_to_date, 1628
test_empty, 1628
test_g_minimized, 1628
test_g_pending, 1628
test_g_up_to_date, 1628
test_sat_c_up_to_date, 1629
test_sat_g_up_to_date, 1629
test_zero_dim_univ, 1629
ZERO_DIM_UNIV, 1630
status
Parma_Polyhedra_Library::BD_Shape, 337
Parma_Polyhedra_Library::Box, 435
Parma_Polyhedra_Library::Grid, 831
Parma_Polyhedra_Library::MIP_Problem, 1130
Parma_Polyhedra_Library::Octagonal_Shape, 1227
Parma_Polyhedra_Library::PIP_Problem, 1320
Parma_Polyhedra_Library::Polyhedron, 1514
std, 241
std::swap
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Congruence_System, 522
stdiobuf
Parma_Polyhedra_Library::stdiobuf, 1651
stdiobuf.cc, 1908
stdiobuf.defs.hh, 1908
stdiobuf.inlines.hh, 1909
stdiobuf.types.hh, 1909
steepest_edge_exact_entering_index
Parma_Polyhedra_Library::MIP_Problem, 1125
steepest_edge_float_entering_index
Parma_Polyhedra_Library::MIP_Problem, 1126
STRICT_INEQUALITY
Parma_Polyhedra_Library::Constraint, 571
strict_subset
Parma_Polyhedra_Library::Bit_Row, 362
strictly_contains
Parma_Polyhedra_Library::BD_Shape, 325
Parma_Polyhedra_Library::Box, 424
Parma_Polyhedra_Library::Grid, 820
Parma_Polyhedra_Library::Interval, 925
Parma_Polyhedra_Library::Octagonal_Shape, 1210
Parma_Polyhedra_Library::Partially_Reduced_Product, 1283
Parma_Polyhedra_Library::Pointset_Powerset, 1392
Parma_Polyhedra_Library::Polyhedron, 1498
STRICTLY_INTERSECTS
Parma_Polyhedra_Library::Poly_Con_Relation, 1407
strictly_intersects
Parma_Polyhedra_Library::Poly_Con_Relation, 1405
strong_closure_assign
Parma_Polyhedra_Library::Octagonal_Shape, 1211
strong_coherence_assign
Parma_Polyhedra_Library::Octagonal_Shape, 1211
strong_normalize
Parma_Polyhedra_Library::Congruence, 499
Parma_Polyhedra_Library::Grid_Generator, 853
Parma_Polyhedra_Library::Linear_Row, 1053
Parma_Polyhedra_Library::Linear_System, 1075
strong_reduction_assign
Parma_Polyhedra_Library::Octagonal_Shape, 1212
STRONGLY_CLOSED
Parma_Polyhedra_Library::Octagonal_Shape::Status, 1648
strongly_minimize_constraints
Parma_Polyhedra_Library::Polyhedron, 1499
strongly_minimize_generators
Parma_Polyhedra_Library::Polyhedron, 1499
strtoll
checked_int.inlines.hh, 1733
strtoull
checked_int.inlines.hh, 1733
sub_2exp_float
Parma_Polyhedra_Library::Checked, 216
sub_2exp_mpq
Parma_Polyhedra_Library::Checked, 216
sub_2exp_mpz
Parma_Polyhedra_Library::Checked, 216
sub_2exp_signed_int

Parma_Polyhedra_Library::Checked, 216
sub_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 216
sub_assign
Parma_Polyhedra_Library::Interval, 925
sub_float
Parma_Polyhedra_Library::Checked, 217
sub_int_larger
Parma_Polyhedra_Library::Checked, 217
sub_mpq
Parma_Polyhedra_Library::Checked, 217
sub_mpz
Parma_Polyhedra_Library::Checked, 217
sub_mul_assign
Parma_Polyhedra_Library, 160
Parma_Polyhedra_Library::Checked_Number, 474
Parma_Polyhedra_Library::GMP_Integer, 749
Parma_Polyhedra_Library::Linear_-Expression, 1038
sub_mul_float
Parma_Polyhedra_Library::Checked, 217
sub_mul_int
Parma_Polyhedra_Library::Checked, 217
sub_mul_mpq
Parma_Polyhedra_Library::Checked, 218
sub_mul_mpz
Parma_Polyhedra_Library::Checked, 218
sub_restriction
Parma_Polyhedra_Library, 160
sub_signed_int
Parma_Polyhedra_Library::Checked, 218
sub_unsigned_int
Parma_Polyhedra_Library::Checked, 218
submod
Parma_Polyhedra_Library, 160
subset_or_equal
Parma_Polyhedra_Library::Bit_Row, 363
SUBSUMES
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1413
subsumes
Parma_Polyhedra_Library::Poly_Gen_-Relation, 1411
succ_float
Parma_Polyhedra_Library::Checked, 218
sum_sign
Parma_Polyhedra_Library::Checked, 218
swap
Parma_Polyhedra_Library, 161
Parma_Polyhedra_Library::BD_Shape, 325, 335
Parma_Polyhedra_Library::Bit_Matrix, 352, 353
Parma_Polyhedra_Library::Bit_Row, 360, 363
Parma_Polyhedra_Library::Box, 424
Parma_Polyhedra_Library::Checked_Number, 474
Parma_Polyhedra_Library::Congruence, 500, 503
Parma_Polyhedra_Library::Congruence_-System, 521, 522
Parma_Polyhedra_Library::Constraint, 580, 586
Parma_Polyhedra_Library::Constraint_-System, 604, 605
Parma_Polyhedra_Library::DB_Matrix, 616, 618
Parma_Polyhedra_Library::DB_Row, 631, 632
Parma_Polyhedra_Library::Determinate, 642, 644
Parma_Polyhedra_Library::Generator, 718, 725
Parma_Polyhedra_Library::Generator_-System, 743, 744
Parma_Polyhedra_Library::Grid, 821, 830
Parma_Polyhedra_Library::Grid_Generator, 853, 856
Parma_Polyhedra_Library::Grid_Generator_-System, 874, 875
Parma_Polyhedra_Library::Interval, 925, 929
Parma_Polyhedra_Library::Interval_Info_-Bitset, 936
Parma_Polyhedra_Library::Interval_Info_-Null, 940
Parma_Polyhedra_Library::Linear_-Expression, 1032, 1038
Parma_Polyhedra_Library::Linear_Row, 1055
Parma_Polyhedra_Library::Linear_System, 1076, 1077
Parma_Polyhedra_Library::Matrix, 1094, 1095
Parma_Polyhedra_Library::MIP_Problem, 1126, 1127
Parma_Polyhedra_Library::Octagonal_Shape, 1212, 1224
Parma_Polyhedra_Library::OR_Matrix, 1241, 1244
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1283, 1287
Parma_Polyhedra_Library::PIP_Problem, 1317, 1318
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 257
Parma_Polyhedra_Library::Pointset_-Powerset, 1393, 1398

Parma_Polyhedra_Library::Polyhedron, 1500, 1511
Parma_Polyhedra_Library::Powerset, 1531, 1532
Parma_Polyhedra_Library::Row, 1566, 1568
swap_columns
Parma_Polyhedra_Library::Matrix, 1094
sync
Parma_Polyhedra_Library::stdiobuf, 1651

t
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1658
T_MAYBE
Parma_Polyhedra_Library, 122
T_NO
Parma_Polyhedra_Library, 122
T_YES
Parma_Polyhedra_Library, 122
Tableau
Parma_Polyhedra_Library::PIP_Solution_-Node::Tableau, 1655
tableau
Parma_Polyhedra_Library::MIP_Problem, 1130
Parma_Polyhedra_Library::PIP_Solution_-Node, 1333
termination.cc, 1910
termination.defs.hh, 1911
termination.templates.hh, 1914
PRINT_DEBUG_INFO, 1917
termination_test_MS
Parma_Polyhedra_Library, 161
Parma_Polyhedra_-Library::Implementation::Termination, 231
termination_test_MS_2
Parma_Polyhedra_Library, 162
termination_test_PR
Parma_Polyhedra_Library, 163
Parma_Polyhedra_-Library::Implementation::Termination, 231
termination_test_PR_2
Parma_Polyhedra_Library, 163
termination_test_PR_original
Parma_Polyhedra_-Library::Implementation::Termination, 231
Ternary
Parma_Polyhedra_Library, 122
test_all
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
Parma_Polyhedra_Library::Box::Status, 1614
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1647
Parma_Polyhedra_-Library::Polyhedron::Status, 1639
Status, 1628
test_any
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
Parma_Polyhedra_Library::Box::Status, 1614
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1648
Parma_Polyhedra_-Library::Polyhedron::Status, 1639
Status, 1628
test_bits
Parma_Polyhedra_Library::Row::Flags, 660
test_c_minimized
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_-Library::Polyhedron::Status, 1640
Status, 1628
test_c_pending
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_-Library::Polyhedron::Status, 1640
Status, 1628
test_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_-Library::Polyhedron::Status, 1640
Status, 1628
test_empty
Parma_Polyhedra_Library::BD_-Shape::Status, 1620
Parma_Polyhedra_Library::Box::Status, 1614
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_Library::Octagonal_-Shape::Status, 1648
Parma_Polyhedra_-Library::Polyhedron::Status, 1640
Status, 1628
test_empty_up_to_date
Parma_Polyhedra_Library::Box::Status, 1614
test_g_minimized
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_-Library::Polyhedron::Status, 1640
Status, 1628
test_g_pending
Parma_Polyhedra_Library::Grid::Status, 1607

Parma_Polyhedra_-
Library::Polyhedron::Status, 1641
Status, 1628
test_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1607
Parma_Polyhedra_-
Library::Polyhedron::Status, 1641
Status, 1628
test_sat_c_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_-
Library::Polyhedron::Status, 1641
Status, 1629
test_sat_g_up_to_date
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_-
Library::Polyhedron::Status, 1641
Status, 1629
test_shortest_path_closed
Parma_Polyhedra_Library::BD_-
Shape::Status, 1620
test_shortest_path_reduced
Parma_Polyhedra_Library::BD_-
Shape::Status, 1621
test_strongly_closed
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1648
test_universe
Parma_Polyhedra_Library::Box::Status, 1614
test_zero_dim_univ
Parma_Polyhedra_Library::BD_-
Shape::Status, 1621
Parma_Polyhedra_Library::Grid::Status, 1608
Parma_Polyhedra_Library::Octagonal_-
Shape::Status, 1648
Parma_Polyhedra_-
Library::Polyhedron::Status, 1641
Status, 1629
textbook_entering_index
Parma_Polyhedra_Library::MIP_Problem,
1126
Three_Valued_Boolean
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Polyhedron, 1431
Threshold
Parma_Polyhedra_Library::Weightwatch_-
Traits, 1677
throw_constraint_incompatible
Parma_Polyhedra_Library::Box, 425
Parma_Polyhedra_Library::Octagonal_Shape,
1212
throw_dimension_incompatible
Parma_Polyhedra_Library::BD_Shape, 325,
326
Parma_Polyhedra_Library::Box, 425, 426
Parma_Polyhedra_Library::Congruence, 500
Parma_Polyhedra_Library::Constraint, 580
Parma_Polyhedra_Library::Generator, 718
Parma_Polyhedra_Library::Grid, 821–823
Parma_Polyhedra_Library::Octagonal_Shape,
1213, 1214
Parma_Polyhedra_Library::Polyhedron, 1500–
1502
throw_expression_too_complex
Parma_Polyhedra_Library::BD_Shape, 327
Parma_Polyhedra_Library::Box, 427
Parma_Polyhedra_Library::Octagonal_Shape,
1214
throw_generic
Parma_Polyhedra_Library::BD_Shape, 327
Parma_Polyhedra_Library::Box, 427
Parma_Polyhedra_Library::Octagonal_Shape,
1214
throw_invalid_argument
Parma_Polyhedra_Library::Congruence, 500
Parma_Polyhedra_Library::Constraint, 580
Parma_Polyhedra_Library::Generator, 719
Parma_Polyhedra_Library::Grid, 823
Parma_Polyhedra_Library::Grid_Generator,
854
Parma_Polyhedra_Library::Polyhedron, 1502
throw_invalid_constraint
Parma_Polyhedra_Library::Grid, 823
throw_invalid_constraints
Parma_Polyhedra_Library::Grid, 824
throw_invalid_generator
Parma_Polyhedra_Library::Grid, 824
Parma_Polyhedra_Library::Polyhedron, 1503
throw_invalid_generators
Parma_Polyhedra_Library::Grid, 824
Parma_Polyhedra_Library::Polyhedron, 1503
throw_me
Parma_Polyhedra_Library::Throwable, 1660
throw_result_exception
Parma_Polyhedra_Library, 163
throw_runtime_error
Parma_Polyhedra_Library::Grid, 824
Parma_Polyhedra_Library::Polyhedron, 1503
throw_space_dimension_overflow
Parma_Polyhedra_Library::Box, 427
Parma_Polyhedra_Library::Grid, 824
Parma_Polyhedra_Library::Polyhedron, 1503
throw_topology_incompatible
Parma_Polyhedra_Library::Polyhedron, 1503,
1504
tight_closure_assign
Parma_Polyhedra_Library::Octagonal_Shape,
1215

tight_coherence_would_make_empty
Parma_Polyhedra_Library::Octagonal_Shape, 1215
time_elapse_assign
Parma_Polyhedra_Library::BD_Shape, 327
Parma_Polyhedra_Library::Box, 427
Parma_Polyhedra_Library::Grid, 824
Parma_Polyhedra_Library::Octagonal_Shape, 1216
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1283
Parma_Polyhedra_Library::Pointset_-Powerset, 1393
Parma_Polyhedra_Library::Polyhedron, 1504
Tmp_Interval_Type
Parma_Polyhedra_Library::Box, 378
To
Parma_Polyhedra_Library::Checked, 223
To1
Parma_Polyhedra_Library::Checked, 224
To2
Parma_Polyhedra_Library::Checked, 224
To3
Parma_Polyhedra_Library::Checked, 224
topological_closure_assign
Parma_Polyhedra_Library::BD_Shape, 328
Parma_Polyhedra_Library::Box, 428
Parma_Polyhedra_Library::Grid, 825
Parma_Polyhedra_Library::Interval, 925
Parma_Polyhedra_Library::Octagonal_Shape, 1216
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1284
Parma_Polyhedra_Library::Pointset_-Powerset, 1393
Parma_Polyhedra_Library::Polyhedron, 1505
Topology
PPL_CXX_interface, 103
topology
Parma_Polyhedra_Library::Linear_Row, 1053
Parma_Polyhedra_Library::Linear_-Row::Flags, 667
Parma_Polyhedra_Library::Linear_System, 1076
Parma_Polyhedra_Library::Polyhedron, 1505
Topology.hh, 1917
Topology_Adjusted_Scalar_Product_Sign
Parma_Polyhedra_Library::Grid_Generator, 856
Parma_Polyhedra_Library::Topology_-Adjusted_Scalar_Product_Sign, 1662
total_memory_in_bytes
Parma_Polyhedra_Library, 163, 164
Parma_Polyhedra_Library::BD_Shape, 328
Parma_Polyhedra_Library::Bit_Matrix, 352
Parma_Polyhedra_Library::Bit_Row, 361
Parma_Polyhedra_Library::Box, 428
Parma_Polyhedra_Library::Checked_Number, 474
Parma_Polyhedra_Library::Congruence, 500
Parma_Polyhedra_Library::Congruence_-System, 521
Parma_Polyhedra_Library::Constraint, 581
Parma_Polyhedra_Library::Constraint_-System, 604
Parma_Polyhedra_Library::DB_Matrix, 616
Parma_Polyhedra_Library::DB_Row, 631
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 896, 897
Parma_Polyhedra_Library::Determinate, 642
Parma_Polyhedra_Library::Determinate::Rep, 1545
Parma_Polyhedra_Library::Generator, 719
Parma_Polyhedra_Library::Generator_-System, 743
Parma_Polyhedra_Library::Grid, 825
Parma_Polyhedra_Library::Grid_Generator, 854
Parma_Polyhedra_Library::Grid_Generator_-System, 874
Parma_Polyhedra_Library::Interval, 926
Parma_Polyhedra_Library::Linear_-Expression, 1032
Parma_Polyhedra_Library::Linear_System, 1076
Parma_Polyhedra_Library::Matrix, 1095
Parma_Polyhedra_Library::MIP_Problem, 1127
Parma_Polyhedra_Library::Octagonal_Shape, 1216
Parma_Polyhedra_Library::OR_Matrix, 1242
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1284
Parma_Polyhedra_Library::PIP_Decision_-Node, 1295
Parma_Polyhedra_Library::PIP_Problem, 1318
Parma_Polyhedra_Library::PIP_Solution_-Node, 1329
Parma_Polyhedra_Library::PIP_Tree_Node, 1344
Parma_Polyhedra_Library::PIP_Tree_-Node::Artificial_Parameter, 258
Parma_Polyhedra_Library::Pointset_-Powerset, 1393
Parma_Polyhedra_Library::Polyhedron, 1506
Parma_Polyhedra_Library::Powerset, 1531
Parma_Polyhedra_Library::Row, 1567

Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 889
Parma_Polyhedra_Library::Variable, 1670
Parma_Polyhedra_Library::Variables_Set, 1675
Traits
Parma_Polyhedra_Library::const_iterator_-to_const, 557
Parma_Polyhedra_Library::iterator_to_const, 993
traits_type
Parma_Polyhedra_Library::stdiobuf, 1650
transpose
Parma_Polyhedra_Library::Bit_Matrix, 352
transpose_assign
Parma_Polyhedra_Library::Bit_Matrix, 352
true_child
Parma_Polyhedra_Library::PIP_Decision_-Node, 1297
trunc_assign
Parma_Polyhedra_Library::Checked_Number, 474
trunc_float
Parma_Polyhedra_Library::Checked, 219
trunc_mpq
Parma_Polyhedra_Library::Checked, 219
TVB_DONT_KNOW
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Polyhedron, 1432
TVB_FALSE
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Polyhedron, 1432
TVB_TRUE
Parma_Polyhedra_Library::Grid, 766
Parma_Polyhedra_Library::Polyhedron, 1432
Type
Parma_Polyhedra_Library::Checked, 224
Parma_Polyhedra_Library::Constraint, 571
Parma_Polyhedra_Library::Generator, 709
Parma_Polyhedra_Library::Grid_Generator, 843
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
type
Parma_Polyhedra_Library::Boundary_Value, 365
Parma_Polyhedra_Library::Boundary_Value< T, typename Enable_If< Is_Same_Or_-Derived< Interval_Base, T >::value >::type >, 366
Parma_Polyhedra_Library::Constraint, 581
Parma_Polyhedra_Library::Enable_If< true, T >, 647
Parma_Polyhedra_Library::Enable_If_Is, 647
Parma_Polyhedra_Library::Generator, 719
Parma_Polyhedra_Library::Grid_Generator, 854
Parma_Polyhedra_Library::Interval_-NS::Property, 1536
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_-If< Is_Native_Or_Checked< From >::value >::type >, 1547
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, type-name From::info_type >::value >::type >, 1548
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Modulo_-Base, typename From::info_type >::value >::type >, 1549
Parma_Polyhedra_Library::Restriction_-Integer< From, Base, typename Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, type-name From::info_type >::value >::type >, 1550
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, typename Enable_If< Is_Native_Or_-Checked< From >::value >::type >, 1551
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, type-name Enable_If< Is_Same_Or_Derived< Interval_Restriction_Integer_Base, type-name From::info_type >::value >::type >, 1552
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, typename Enable_If< Is_Same_Or_-Derived< Interval_Restriction_Integer_-Modulo_Base, typename From::info_type >::value >::type >, 1553
Parma_Polyhedra_Library::Restriction_-Integer_Modulo< From, T, Base, type-name Enable_If< Is_Same_Or_Derived< Interval_Restriction_None_Base, type-name From::info_type >::value >::type >, 1554
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type, 1585
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< char >, 1586

Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< float >, 1586
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed char >, 1587
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed int >, 1588
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed long >, 1588
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< signed short >, 1589
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned char >, 1589
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned int >, 1590
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned long >, 1590
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned long long >, 1591
Parma_Polyhedra_Library::Select_Temp_-Boundary_Type< unsigned short >, 1591
type_for_add
Parma_Polyhedra_Library::Checked::Larger< char >, 998
Parma_Polyhedra_Library::Checked::Larger< signed char >, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1001
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1003
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1004
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1006
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1008
Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1009
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1011
Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1012
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1014
type_for_mul
Parma_Polyhedra_Library::Checked::Larger< char >, 998
Parma_Polyhedra_Library::Checked::Larger< signed char >, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1001
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1003
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1004
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1006
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1008
Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1009
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1011
Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1012
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1014
type_for_neg
Parma_Polyhedra_Library::Checked::Larger< char >, 998
Parma_Polyhedra_Library::Checked::Larger< signed char >, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1001
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1003
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1005
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1006
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1008
Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1009
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1011
Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1013
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1014
type_for_sub
Parma_Polyhedra_Library::Checked::Larger< char >, 998
Parma_Polyhedra_Library::Checked::Larger< signed char >, 1000
Parma_Polyhedra_Library::Checked::Larger< signed int >, 1001
Parma_Polyhedra_Library::Checked::Larger< signed long >, 1003
Parma_Polyhedra_Library::Checked::Larger< signed long long >, 1005
Parma_Polyhedra_Library::Checked::Larger< signed short >, 1006
Parma_Polyhedra_Library::Checked::Larger< unsigned char >, 1008

Parma_Polyhedra_Library::Checked::Larger< unsigned int >, 1009
Parma_Polyhedra_Library::Checked::Larger< unsigned long >, 1011
Parma_Polyhedra_Library::Checked::Larger< unsigned long long >, 1013
Parma_Polyhedra_Library::Checked::Larger< unsigned short >, 1014

uflow
Parma_Polyhedra_Library::stdiobuf, 1651
uint_fast16_t
Parma_Polyhedra_Library::Checked, 178
uint_fast32_t
Parma_Polyhedra_Library::Checked, 178
uint_fast64_t
Parma_Polyhedra_Library::Checked, 178
umod_2exp_float
Parma_Polyhedra_Library::Checked, 219
umod_2exp_mpq
Parma_Polyhedra_Library::Checked, 219
umod_2exp_mpz
Parma_Polyhedra_Library::Checked, 219
umod_2exp_signed_int
Parma_Polyhedra_Library::Checked, 219
umod_2exp_unsigned_int
Parma_Polyhedra_Library::Checked, 220
UNBOUNDED
Parma_Polyhedra_Library::MIP_Problem, 1106
UNBOUNDED_MIP_PROBLEM
PPL_CXX_interface, 102
unconstrain
Parma_Polyhedra_Library::BD_Shape, 328, 329
Parma_Polyhedra_Library::Box, 428
Parma_Polyhedra_Library::Grid, 825
Parma_Polyhedra_Library::Octagonal_Shape, 1216, 1217
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1284
Parma_Polyhedra_Library::Pointset_-Powerset, 1394
Parma_Polyhedra_Library::Polyhedron, 1506
underflow
Parma_Polyhedra_Library::stdiobuf, 1652
UNFEASIBLE_MIP_PROBLEM
PPL_CXX_interface, 102
UNFEASIBLE_PIP_PROBLEM
PPL_CXX_interface, 103
ungetc_buf
Parma_Polyhedra_Library::stdiobuf, 1653
union_helper
Parma_Polyhedra_Library::Bit_Row, 361
UNIVERSE
Parma_Polyhedra_Library::Box::Status, 1615
PPL_CXX_interface, 102
UNKNOWN
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
UNSATISFIABLE
Parma_Polyhedra_Library::MIP_Problem, 1106
Parma_Polyhedra_Library::PIP_Problem, 1309
unset_pending_rows
Parma_Polyhedra_Library::Grid_Generator_-System, 874
Parma_Polyhedra_Library::Linear_System, 1076
UNSIGNED
PPL_CXX_interface, 101
unsupported_value
Parma_Polyhedra_Library::Interval_-NS::Property, 1536
update_congruences
Parma_Polyhedra_Library::Grid, 826
update_constraints
Parma_Polyhedra_Library::Polyhedron, 1507
update_generators
Parma_Polyhedra_Library::Grid, 826
Parma_Polyhedra_Library::Polyhedron, 1507
update_sat_c
Parma_Polyhedra_Library::Polyhedron, 1507
update_sat_g
Parma_Polyhedra_Library::Polyhedron, 1508
update_solution
Parma_Polyhedra_Library::PIP_Solution_-Node, 1330
update_tableau
Parma_Polyhedra_Library::PIP_Decision_-Node, 1296
Parma_Polyhedra_Library::PIP_Solution_-Node, 1330
Parma_Polyhedra_Library::PIP_Tree_Node, 1345
upper
Parma_Polyhedra_Library::Interval, 926
upper_
Parma_Polyhedra_Library::Interval, 929
upper_bound_assign
Parma_Polyhedra_Library::BD_Shape, 329
Parma_Polyhedra_Library::Box, 429
Parma_Polyhedra_Library::Determinate, 643
Parma_Polyhedra_Library::Grid, 827
Parma_Polyhedra_Library::Octagonal_Shape, 1217

Parma_Polyhedra_Library::Partially_-Reduced_Product, 1285
Parma_Polyhedra_Library::Polyhedron, 1508
Parma_Polyhedra_Library::Powerset, 1531
upper_bound_assign_if_exact
Parma_Polyhedra_Library::BD_Shape, 329
Parma_Polyhedra_Library::Box, 429
Parma_Polyhedra_Library::C_Polyhedron, 443
Parma_Polyhedra_Library::Grid, 827
Parma_Polyhedra_Library::NNC_Polyhedron, 1142
Parma_Polyhedra_Library::Octagonal_Shape, 1218
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1285
Parma_Polyhedra_Library::Powerset, 1531
upper_constraint
Parma_Polyhedra_Library::Interval, 927
upper_extend
Parma_Polyhedra_Library::Interval, 927
upper_is_boundary_infinity
Parma_Polyhedra_Library::Interval, 927
upper_is_domain_sup
Parma_Polyhedra_Library::Interval, 928
upper_is_open
Parma_Polyhedra_Library::Interval, 928
upper_normalize
Parma_Polyhedra_Library::Interval, 928
upper_triangular
Parma_Polyhedra_Library::Grid, 827
USE_BFT00
C_Polyhedron.cc, 1701
USE_BHZ09
C_Polyhedron.cc, 1701
NNC_Polyhedron.cc, 1856

v
Parma_Polyhedra_Library::Checked_Number, 475
Value
Parma_Polyhedra_Library::Interval_-NS::Property, 1535
value
Parma_Polyhedra_Library::Bool, 365
Parma_Polyhedra_Library::Fit< T, v, type-name Enable_If< C_Integer< T >::value >::type >, 655
Parma_Polyhedra_Library::Is_Same_Or_-Derived, 982
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 251
Parma_Polyhedra_Library::TConstant, 1659
value_type
Parma_Polyhedra_Library::const_iterator_-to_const, 557
Parma_Polyhedra_Library::DB_-Matrix::const_iterator, 531
Parma_Polyhedra_Library::iterator_to_const, 993
Parma_Polyhedra_Library::Matrix::const_-iterator, 527
Parma_Polyhedra_Library::OR_Matrix::any_-row_iterator, 245
Parma_Polyhedra_Library::Pointset_-Powerset, 1359
Parma_Polyhedra_Library::Powerset, 1521
var_column
Parma_Polyhedra_Library::PIP_Solution_-Node, 1333
var_row
Parma_Polyhedra_Library::PIP_Solution_-Node, 1333
Variable
Parma_Polyhedra_Library::Variable, 1667
Variable.cc, 1918
Variable.defs.hh, 1918
Variable.inlines.hh, 1919
Variable.types.hh, 1920
Variables_Set
Parma_Polyhedra_Library::Variables_Set, 1673
Variables_Set.cc, 1920
Variables_Set.defs.hh, 1921
Variables_Set.inlines.hh, 1922
Variables_Set.types.hh, 1923
varid
Parma_Polyhedra_Library::Variable, 1671
vclass
Parma_Polyhedra_Library::Minus_Infinity, 1098
Parma_Polyhedra_Library::Not_A_Number, 1145
Parma_Polyhedra_Library::Plus_Infinity, 1347
vec
Parma_Polyhedra_Library::Bit_Row, 363
Parma_Polyhedra_Library::OR_Matrix, 1244
vec_
Parma_Polyhedra_Library::DB_Row_Impl_-Handler::Impl, 897
Parma_Polyhedra_Library::Row_Impl_-Handler::Impl, 889
vec_capacity
Parma_Polyhedra_Library::OR_Matrix, 1245
version
Parma_Polyhedra_Library, 164
version.cc, 1923

version.hh, 1924
version_beta
Parma_Polyhedra_Library, 164
version_major
Parma_Polyhedra_Library, 164
version_minor
Parma_Polyhedra_Library, 164
version_revision
Parma_Polyhedra_Library, 165

w_info
Parma_Polyhedra_Library::Interval, 928
w_method
Parma_Polyhedra_Library::Widening_-Function, 1680
weakening_assign
Parma_Polyhedra_Library::Determinate, 643
weight
Parma_Polyhedra_Library::Weightwatch_-Traits, 1678
WEIGHT_ADD
globals.defs.hh, 1791
WEIGHT_ADD_MUL
globals.defs.hh, 1791
WEIGHT_BEGIN
globals.defs.hh, 1792
widen_fun_ref
Parma_Polyhedra_Library::Pointset_-Powerset, 1398
widening_assign
Parma_Polyhedra_Library::BD_Shape, 330
Parma_Polyhedra_Library::Box, 429
Parma_Polyhedra_Library::Grid, 827
Parma_Polyhedra_Library::Octagonal_Shape, 1218
Parma_Polyhedra_Library::Partially_-Reduced_Product, 1285
Parma_Polyhedra_Library::Polyhedron, 1508
Widening_Function
Parma_Polyhedra_Library::Widening_-Function, 1679
Widening_Function.defs.hh, 1925
Widening_Function.inlines.hh, 1925
Widening_Function.types.hh, 1926
Widening_Method
Parma_Polyhedra_Library::Widening_-Function, 1679
word
Parma_Polyhedra_Library::float_ibm_single, 675
Parma_Polyhedra_Library::float_ieee754_-half, 685
Parma_Polyhedra_Library::float_ieee754_-single, 694
working_cost
Parma_Polyhedra_Library::MIP_Problem, 1131
wrap_assign
Parma_Polyhedra_Library::BD_Shape, 330
Parma_Polyhedra_Library::Box, 430
Parma_Polyhedra_Library::Grid, 828
Parma_Polyhedra_Library::Interval, 929
Parma_Polyhedra_Library::Octagonal_Shape, 1218
Parma_Polyhedra_Library::Pointset_-Powerset, 1394
Parma_Polyhedra_Library::Polyhedron, 1509

xsgetn
Parma_Polyhedra_Library::stdiobuf, 1652
xsputn
Parma_Polyhedra_Library::stdiobuf, 1652

ZERO
Parma_Polyhedra_Library::PIP_Solution_-Node, 1324
zero
Parma_Polyhedra_Library::Linear_-Expression, 1032
zero_dim_closure_point
Parma_Polyhedra_Library::Generator, 719
zero_dim_closure_point_p
Parma_Polyhedra_Library::Generator, 725
zero_dim_empty
Parma_Polyhedra_Library::Congruence_-System, 521
Parma_Polyhedra_Library::Constraint_-System, 604
zero_dim_empty_p
Parma_Polyhedra_Library::Congruence_-System, 522
Parma_Polyhedra_Library::Constraint_-System, 606
zero_dim_false
Parma_Polyhedra_Library::Congruence, 500
Parma_Polyhedra_Library::Constraint, 581
zero_dim_false_p
Parma_Polyhedra_Library::Congruence, 503
Parma_Polyhedra_Library::Constraint, 587
zero_dim_integrality
Parma_Polyhedra_Library::Congruence, 501
zero_dim_integrality_p
Parma_Polyhedra_Library::Congruence, 504
zero_dim_point
Parma_Polyhedra_Library::Generator, 720
Parma_Polyhedra_Library::Grid_Generator, 854
zero_dim_point_p

Parma_Polyhedra_Library::Generator, 725
Parma_Polyhedra_Library::Grid_Generator, 857
zero_dim_positivity
Parma_Polyhedra_Library::Constraint, 581
zero_dim_positivity_p
Parma_Polyhedra_Library::Constraint, 587
ZERO_DIM_UNIV
Parma_Polyhedra_Library::BD_Shape::Status, 1621
Parma_Polyhedra_Library::Grid::Status, 1609
Parma_Polyhedra_Library::Octagonal_Shape::Status, 1649
Parma_Polyhedra_Library::Polyhedron::Status, 1643
Status, 1630
zero_dim_univ
Parma_Polyhedra_Library::Generator_System, 743
Parma_Polyhedra_Library::Grid_Generator_System, 874
zero_dim_univ_p
Parma_Polyhedra_Library::Generator_System, 744
Parma_Polyhedra_Library::Grid_Generator_System, 876
zero_p
Parma_Polyhedra_Library::Linear_Expression, 1039