The PPL: A Library for Representing Numerical Abstractions: Current and Future Plans

Roberto BAGNARA, Elena MAZZI, Barbara QUARTIERI, Enea ZAFFANELLA
Katy DOBSON, Patricia M. HILL

University of Parma, Italy
University of Leeds, United Kingdom

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

Plan of the Talk

The Problem

- ➜ Programs need to be designed, developed and maintained over their entire lifespan (up to 20 and more years) at reasonable costs;
- ➜ programs have exploded in size over the last 25 years so that more and more with tens of millions of lines of code are in general use;
- ➜ unassisted development and maintenance teams do not stand a chance to follow such an explosion in size and complexity;
- ➜ the large number of bugs in much of our software is hardly bearable even in office applications. . .
  - ➜ . . . no safety critical application can tolerate this failure rate;
- ➜ the problem of software reliability is one of the most important problems computer science has to face;
- ➜ this justifies the growing interest in mechanical tools to help the programmer reasoning about programs.

Recent News I

[...] The Mars Climate Orbiter burned in the martian atmosphere in 1999 after missing its orbit insertion because unit computations were inconsistent.

The same year, Mars Polar Lander is suspected of having crashed on Mars upon landing when a software flag was not reset properly.

In [. . .] the 1997 Mars Pathfinder (MPF) technology demonstration mission [. . .] a day's exploration time was lost when ground support teams were forced to reboot the system while downloading science data.

[. . .]

NASA's 2003 Mars Exploration Rover (MER) mission includes two rovers [. . .] At a cost of $400 million for each rover, a coding error that shuts down a rover overnight would in effect be a $4.4 million mistake, as well as a loss of valuable exploration time on the planet.

http://www.arc.nasa.gov/exploringtheuniverse-computercheck.cfm

Recent News II

On Aug 14 2003, 50 ml people in the US and Canada had no electricity.

A number of factors and failings came together to make the blackout the worst outage in North American history. One of them was buried in a massive piece of software compiled from four million lines of C code [. . .].

Sometimes working late into the night and the early hours of the morning, the team pored over the approximately one-million lines of code that comprise the XA/21's Alarm and Event Processing Routine, written in the C and C++ programming languages. "This fault was so deeply embedded, it took them weeks of poring through millions of lines of code and data to find it." FE spokesman Ralph DiNicola said.

We had in excess of three million online operational hours in which nothing had ever exercised that bug. I'm not sure that more testing would have revealed it. – GE Energy's Mike Unum [. . .]

http://www.securityfocus.com/news/8412

An Example: Is `x/(x-y)` Well-Defined?

Many things may go wrong

- ➜ `x` and/or `y` may be uninitialized;
- ➜ `x-y` may overflow;
- ➜ `x` and `y` may be equal (or `x-y` may overflow): division by 0;
- ➜ `x/(x-y)` may overflow (or underflow).

What can we do about it?

- ➜ full verification is undecidable;
- ➜ code review: complex, expensive and with volatile results;
- ➜ dynamic testing plus debugging: complex, expensive, does not scale (the cost of testing goes as the square of the program size), but it is repeatable;
- ➜ formal methods: complex and expensive but reusable, can be very thorough, repeatable, scale up to a certain program size then become unapplicable (we are working to extend that limit).

Limit vs. Distribution Information

Most numerical abstract domains are meant to describe limit information, i.e., bounds within which the values must lie

- ➜ intuitively based on convex-set approximation
  - ➜ non-relational: bounding boxes;
  - ➜ relational: bounded differences, octagons, convex polyhedra.

But convex-set based approximations may be too coarse and fail to capture distribution information. . .

- ➜ . . . so we need alternative approximations such as:
  - ➜ a small number of mainly irregular convex-set approximations;
  - ➜ a large (even infinite) number of repetitive occurrences of a single pattern to represent distribution information;
  - ➜ combinations of the two things above.

CONVEX POLYHEDRA: WHAT AND WHY

What?

- ➜ regions of $\mathbb{R}^n$ bounded by a finite set of hyperplanes.

Why? Solving Classical Data-Flow Analysis Problems!

- ➜ array bound checking and compile-time overflow detection;
- ➜ loop invariant computations and loop induction variables.

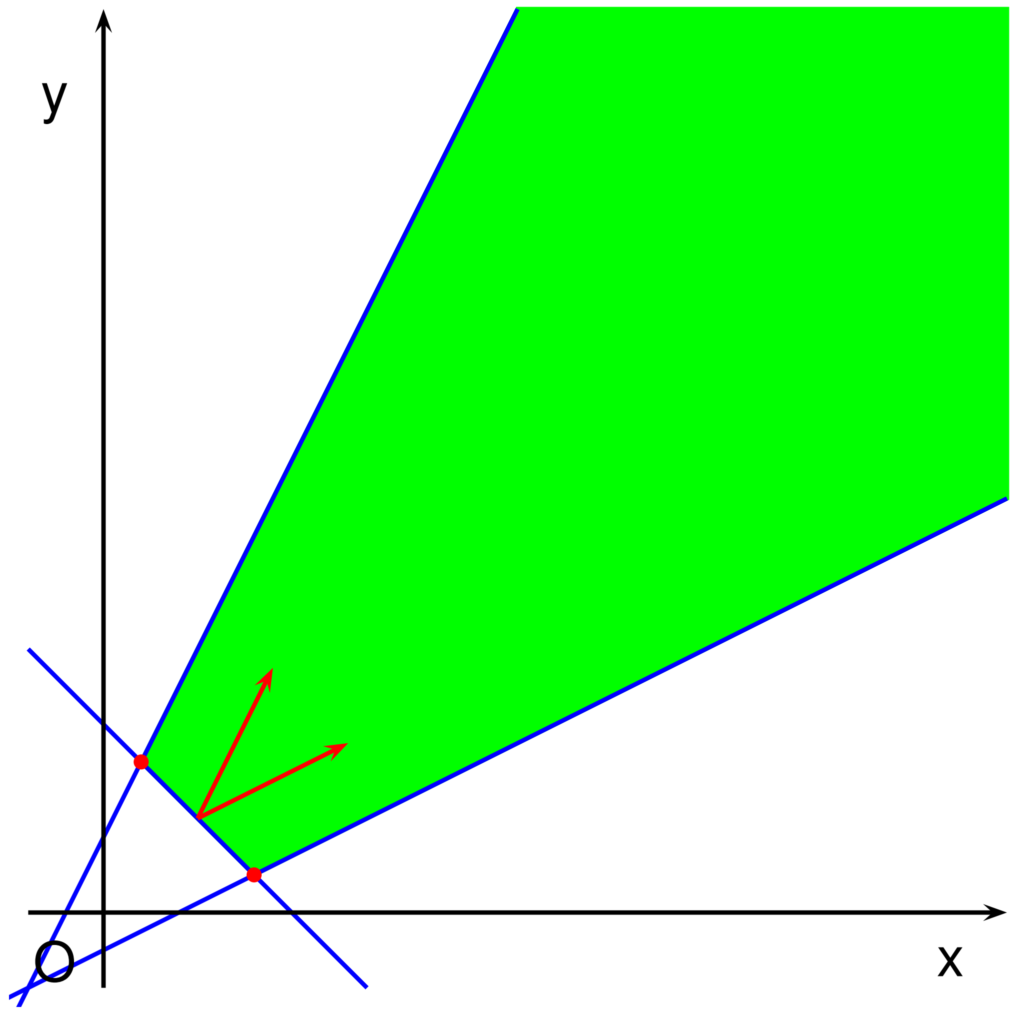
Why? Verification of Concurrent and Reactive Systems!

- ➜ synchronous languages;
- ➜ linear hybrid automata (roughly, FSMs with time requirements);
- ➜ systems based on temporal specifications.

And Again: Many Other Applications. . .

- ➜ inferring argument size relationships in logic programs;
- ➜ termination inference for Prolog programs;
- ➜ string cleanness for C programs.

Example: Constraint and Generator Descriptions



$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

$$\begin{cases} \text{points: } \{(4,1),(1,4)\} \\ \text{rays: } \{(1,2),(2,1)\} \end{cases}$$

The Parma Polyhedra Library

- A collaborative project started in January 2001 at the Department of Mathematics of the University of Parma.
  - The University of Leeds (UK) is now a major contributor to the library.
- It aims at becoming a truly professional library for the handling (not necessarily closed) rational convex polyhedra. We are almost there.
- Targeted at abstract interpretation and computer-aided verification.
- Free software released under the GNU General Public License.

Why yet another library? Some limitations of existing ones:

- data-structures employed cannot grow/shrink dynamically;
- possibility of overflow, underflow and rounding errors;
- unsuitable mechanisms for error detection, handling and recovery;
  - (cannot reliably resume computation with an alternative method, e.g., by reverting to an interval-based approximation).
- Several existing libraries are free, but they do not provide adequate documentation for the interfaces and the code.

Parma Polyhedra Library: Where We Are

PPL 0.5 (released on April 28, 2003)

- ➜ Best available support for both closed and not necessarily closed (NNC) convex polyhedra.
- ➜ A new widening operator that is always more precise than the standard widening.
- ➜ Clean, safe and natural interfaces for C, C++ and Prolog.
- ➜ Comprehensive documentation for both users and developers.
- ➜ Users (VERIMAG, $\mathrm{CMU} \times 2$, ENS Cachan, UMich (Spin), UWisc, ...) are quite satisfied with the performance.

Parma Polyhedra Library: Where Are We Going

PPL 0.6 (to be released by the end of July 2004)

➜ Generic support for finite powerset domains, including widening operators.

➜ Instantiation on powersets of polyhedra: includes the first proposals of non-trivial widenings for this domain.

➜ Summarization operators, as proposed by Gopan et al. at TACAS'04.

PPL 0.7 (to be released by the end of 2004)

➜ An implementation of the simplex algorithm.

➜ Support the use of native integers with overflow detection.

➜ Serialization and deserialization operators.

➜ A preliminary implementation for bounded differences and octagons.

➜ New OCaml interface.

➜ . . .

Parma Polyhedra Library: Where Will We Go

PPL 0.8, 0.9, 0.10, . . . (release dates unknown)

➜ Support for intervals and boxes.

➜ New implementations, based on intervals, for bounded differences, octagons, bounded quotients and other numeric abstractions.

➜ Support for grids and $\mathbb{Z}$-polyhedra.

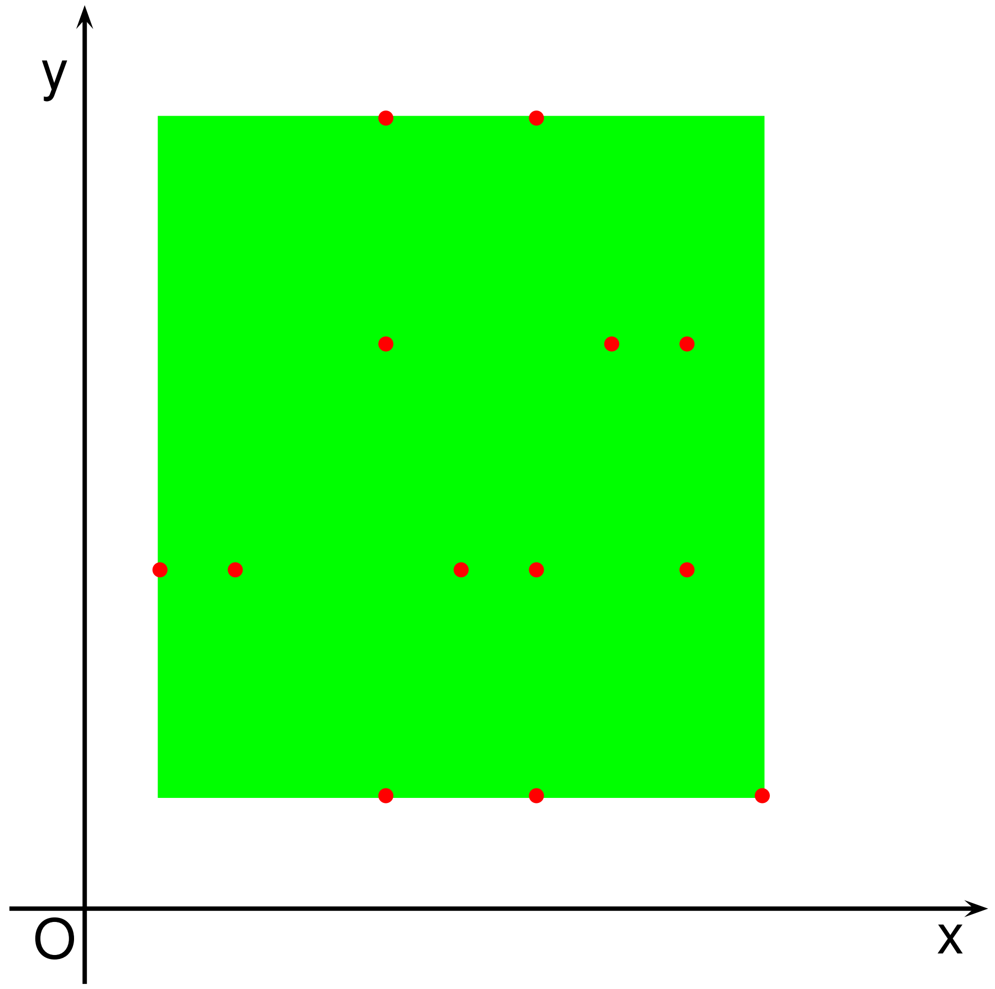
➜ Generic support for ask-and-tell domains.

➜ Cartesian factoring, as proposed by Halbwachs et al. at SAS’03.

➜ New Java interface.

➜ . . .

INTERVALS AND BOUNDING BOXES



$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \end{cases}$$

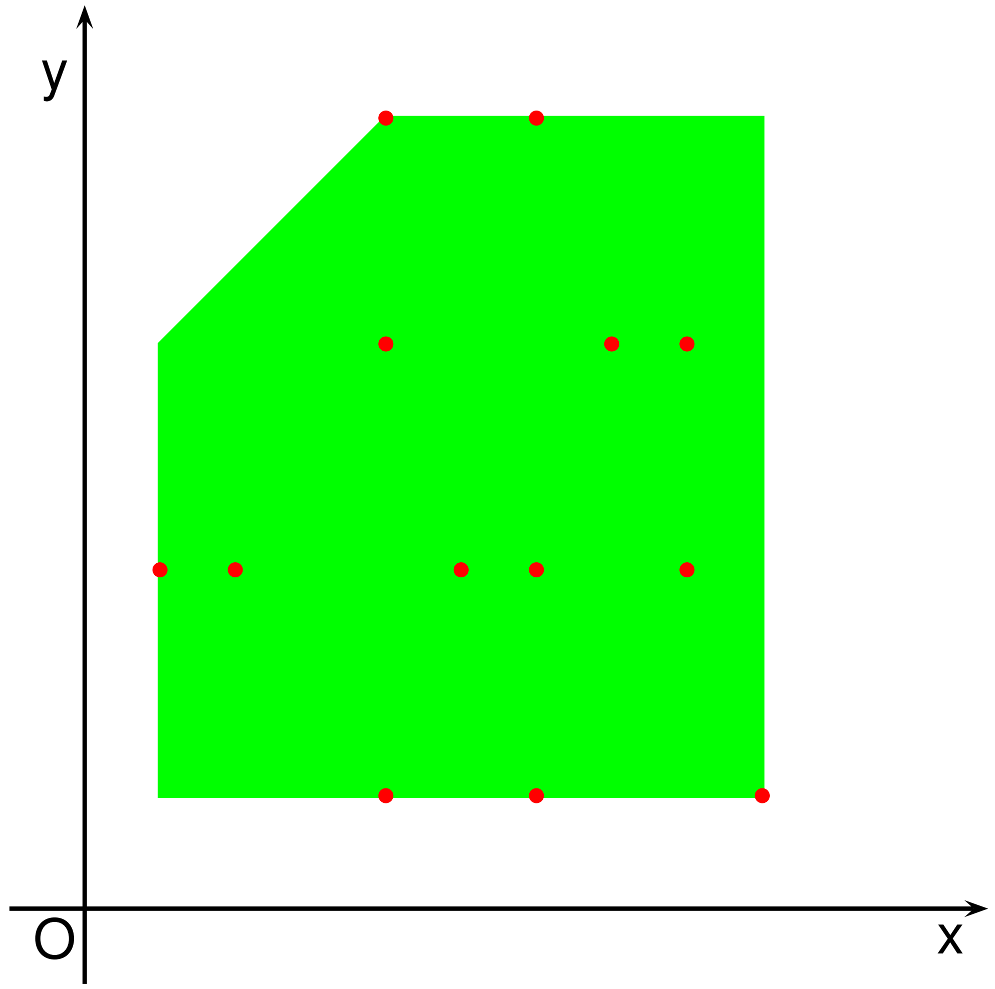
INTERVALS

- ➜ One of the first numerical abstract domains ever proposed (Cousot & Cousot, 1976).
- ➜ The only one with a real scalability guarantee (linear complexity in the number of variables).
- ➜ Useful as a fall-back domain and for the realization of more complex domains.
- ➜ Despite this no available implementation is really suitable for the purposes of abstract interpretation:
  - ➜ lack of support for non-closed intervals (cannot represent constraints of the form $\pm x < b$); and/or
  - ➜ limited choice of the data type used to represent boundaries; and/or
  - ➜ unsound implementation (use floating point numbers to represent the boundaries disregarding rounding errors).

INTERVALS, CONT.

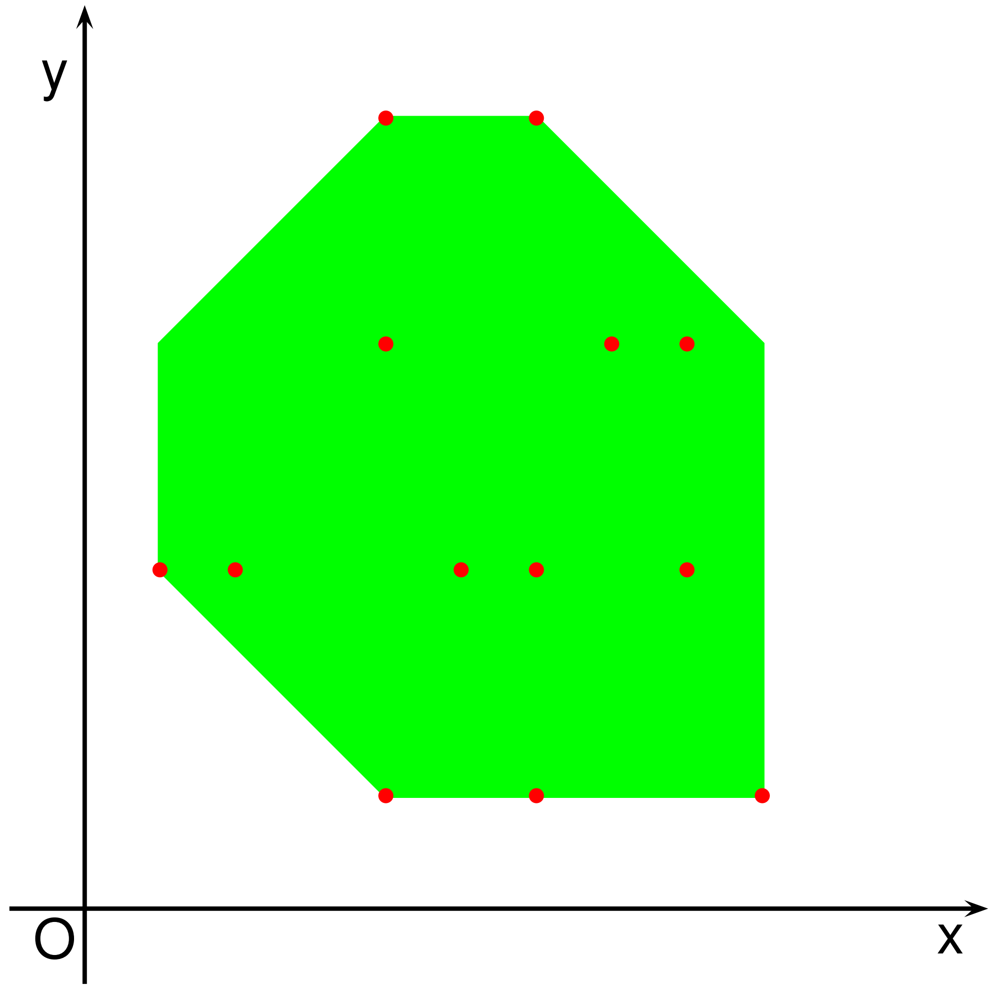
- ➜ Work has already started on a brand new implementation of intervals, especially targeted at the needs of abstract interpretation:
  - ➜ support closed as well as non-closed intervals;
  - ➜ provide a wide selection of number families for the representation of the boundaries; initially, native integers (8, 16, 32 and 64 bits wide), native floats (32, 64 and 128 bits wide), unlimited precision integers and rationals;
  - ➜ support for different controlled rounding strategies (ensuring various degrees of portability and efficiency);
  - ➜ support for both intervals of real numbers and intervals of integer numbers (independently from the type of the boundaries).

Bounded Differences



$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \end{cases}$$

Octagons



$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \\ 11 \leq x + y \leq 33 \end{cases}$$

Bounded Differences, Octagons and Beyond

- ➜ Initial implementations are almost finished:
  - ➜ for bounded differences, a standard implementation based on difference bound matrices is finished and ready to be incorporated into the PPL;
  - ➜ for intervals, an implementation based on papers of Miné is almost finished.
- ➜ These implementations have several limitations:
  - ➜ they only supports closed polyhedra;
  - ➜ lack a satisfactory widening operator.
- ➜ New implementations will be based on intervals.
- ➜ Beyond: the interval abstraction can be used as the basis of other numerical abstractions, such as bounded quotients (Bagnara, 1997).

Limit vs. Distribution Information

Most numerical abstract domains are meant to describe limit information, i.e., bounds within which the values must lie

- ➜ intuitively based on convex-set approximation
  - ➜ non-relational: bounding boxes;
  - ➜ relational: bounded differences, octagons, convex polyhedra.

But convex-set based approximations may be too coarse and fail to capture distribution information. . .

- ➜ . . . so we need alternative approximations such as:
  - ➜ a small number of mainly irregular convex-set approximations;
  - ➜ a large (even infinite) number of repetitive occurrences of a single pattern to represent distribution information;
  - ➜ combinations of the two things above.

Dealing With a Few Irregular Approximations

In this case, the finite powerset construction may be an appropriate approximation domain:

- ➜ having a small number of elements enables the tuning of the efficiency/precision tradeoff;
- ➜ each element in the finite collection can be approximated independently from the other elements.

Dealing With Repetition of a Single Pattern

In this case, distribution information is more important than limit information:

- ➜ linear domains representing discrete and linearly repetitive values
  - ➜ the integer lattice...
  - ➜ integral lattices...
  - ➜ or, more generally, a domain $\mathbb{G}$ of grids.

Combining Limit and Distribution Information

Generalize and/or specialize the work on $\mathbb{Z}$-polyhedra

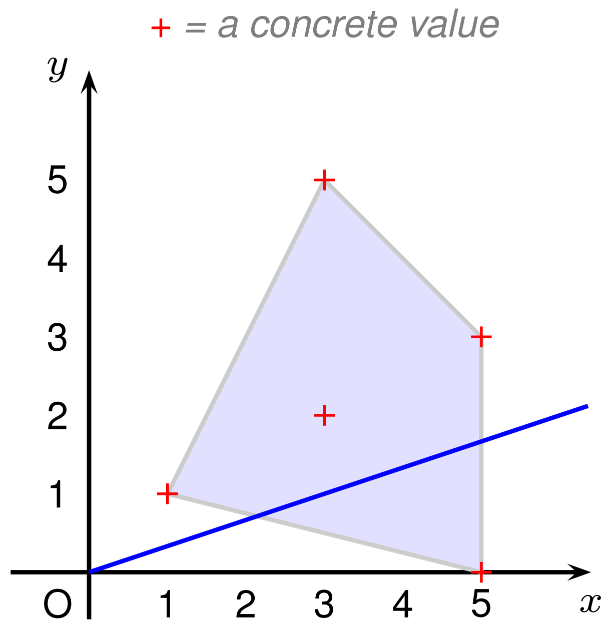
- $\mathbb{G}$-$\star$
- finite powerset of $\mathbb{G}$-$\star$

where $\star$ may be

- bounding boxes,
- bounded differences,
- octagons,
- (convex) polyhedra,
- . . .

GRID: A SIMPLE EXAMPLE IN 2D

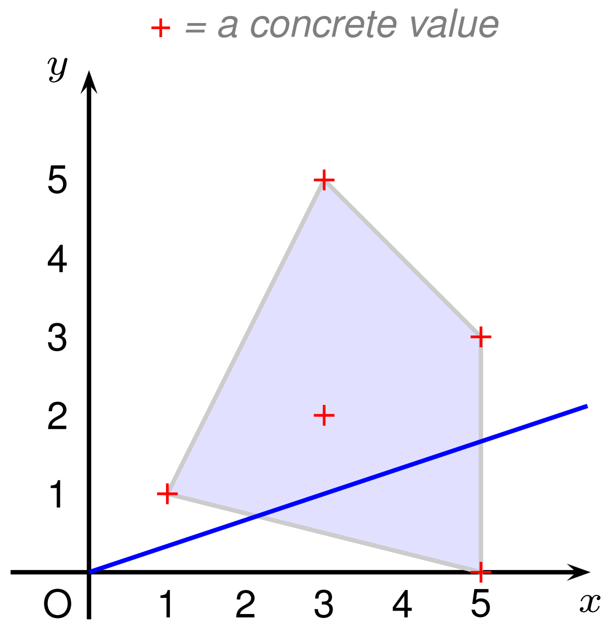
In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



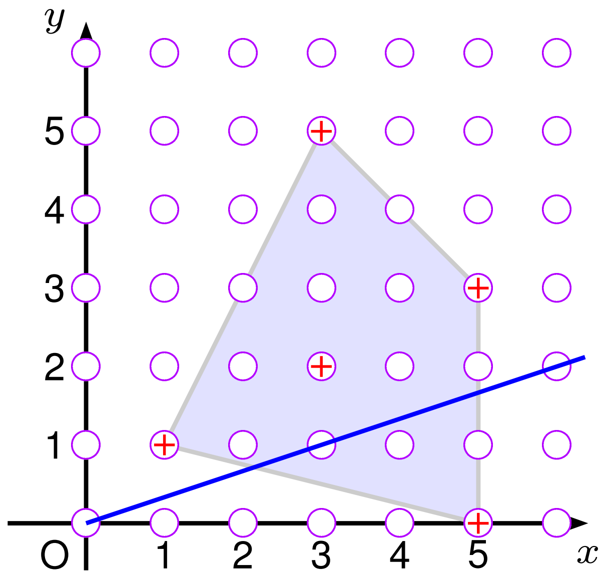
Concrete Domain

GRID: A SIMPLE EXAMPLE IN 2D

In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



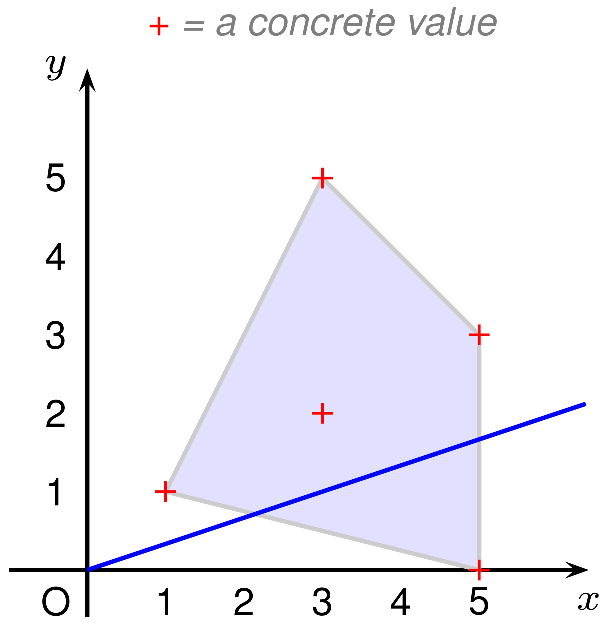
Concrete Domain



An Integer Grid

Grid: A simple example in 2D

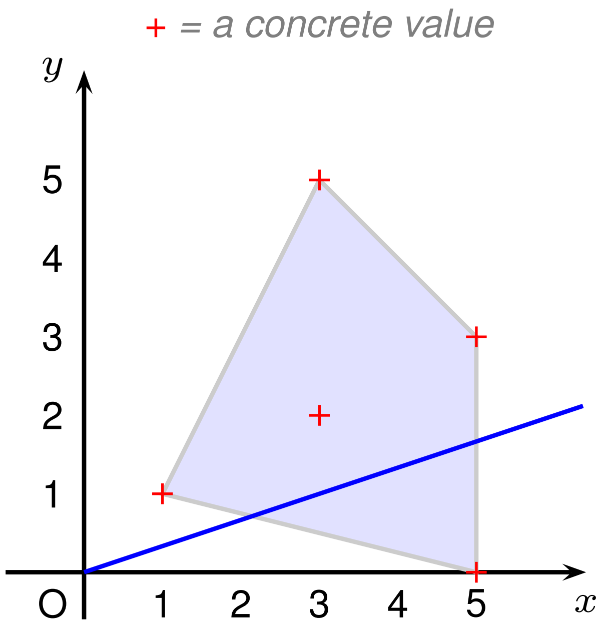
In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



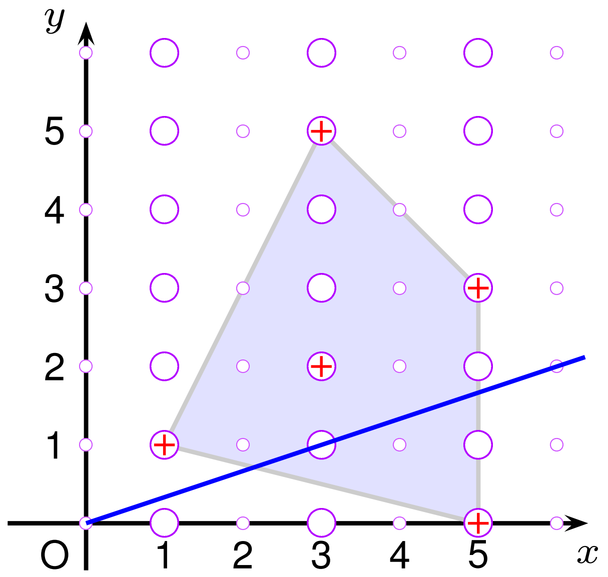
Concrete Domain

Grid: A simple example in 2D

In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



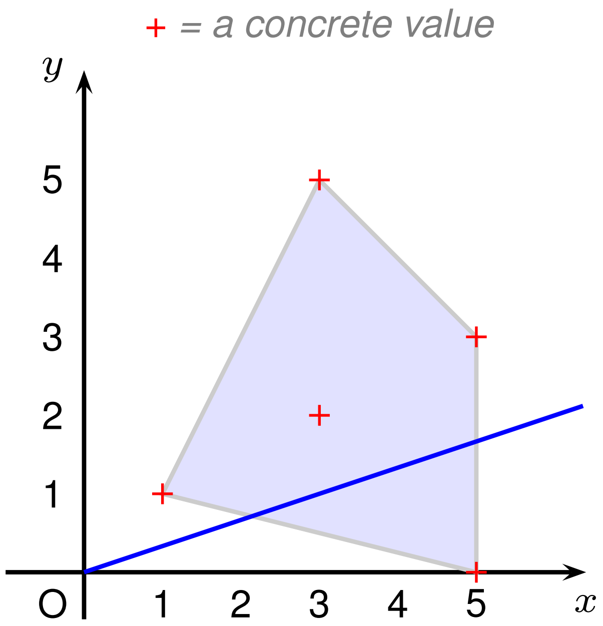
Concrete Domain



A Non-relational Integral Grid

Grid: A simple example in 2D

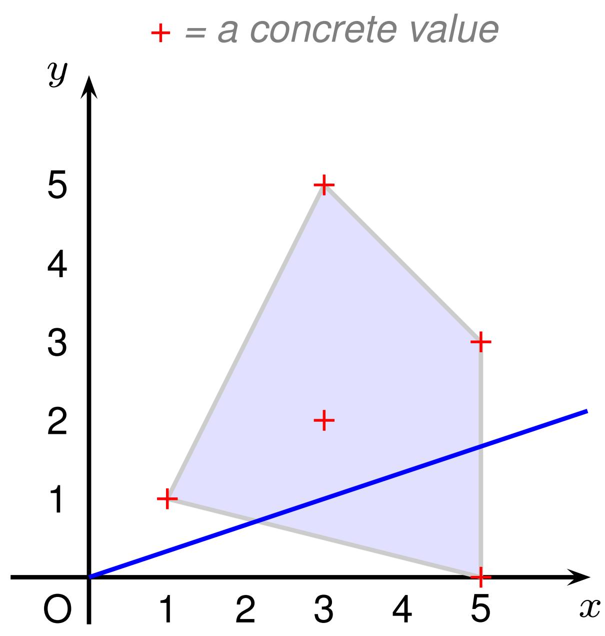
In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



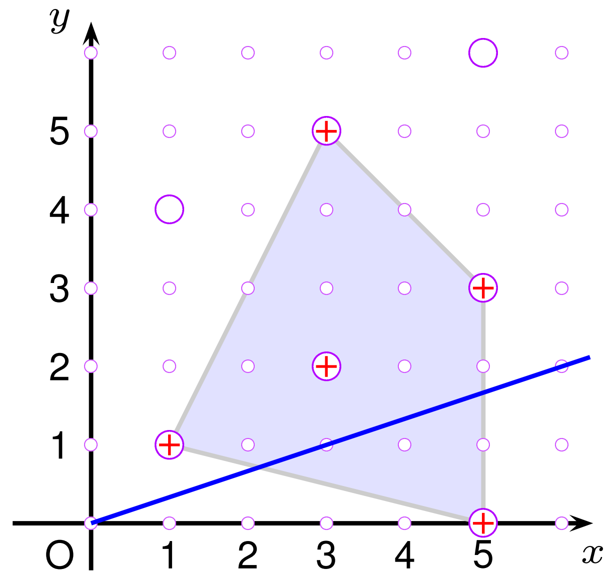
Concrete Domain

GRID: A SIMPLE EXAMPLE IN 2D

In a computation, where x, y are declared as integers, we want to verify that $x - 3y$ is never zero.



Concrete Domain



A Relational Integral Grid

A Program and a Grid Represention

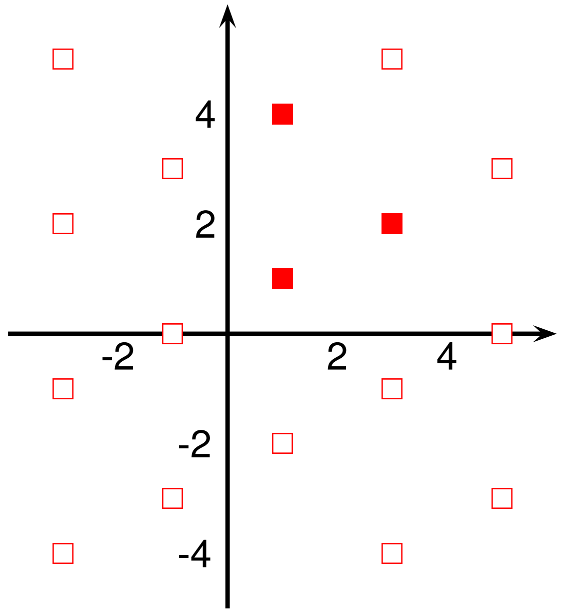
```
x := 2x + 1;
y := 3y - x - 1;
for i := 1 to m
  y := y + 2 - x;
  x := x + 2
endfor
```

Then, at the start of each `for` loop, `x` and `y` can take values

$$\begin{array}{l}
\ldots,(-3,-3),(-3,-1),(-3,\ \ 2),(-3,\ \ 5),\ldots\\
\ldots,(-1,-3),(-1,\ \ 0),(-1,\ \ 3),(-1,\ \ 6),\ldots\\
\ldots,(\ \ 1,-2),(\ \ 1,\ \ 1),(\ \ 1,\ \ 4),(\ \ 1,\ \ 7),\ldots\\
\ldots,(\ \ 3,-1),(\ \ 3,\ \ 2),(\ \ 3,\ \ 5),(\ \ 3,\ \ 8),\ldots
\end{array}$$

Grid Represention: Generators and Congruences

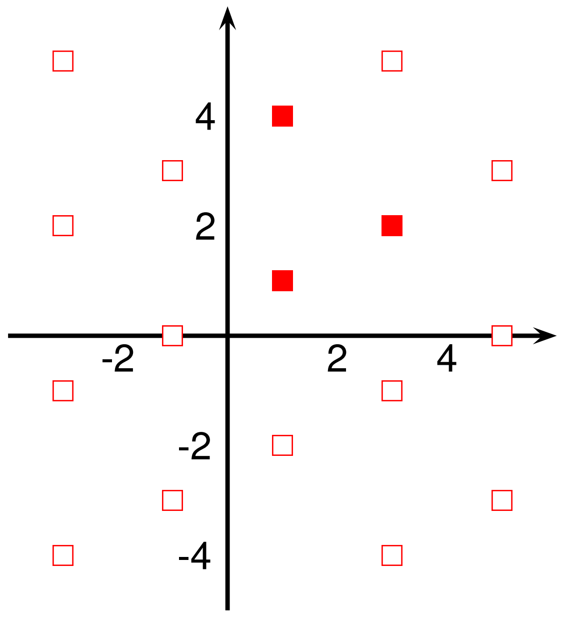
This grid can be represented by a set of generating points or congruence relations:

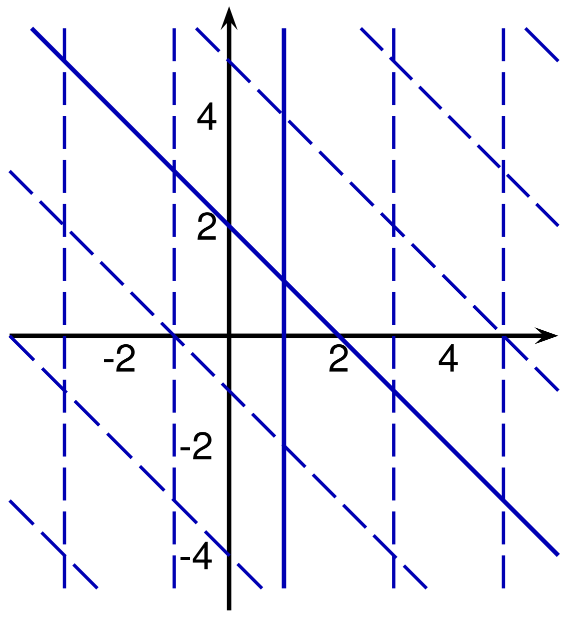


Generators: $\{(1,1)^{\mathrm{T}}, (1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$

Grid Representation: Generators and Congruences

This grid can be represented by a set of generating points or congruence relations:





Generators: $\{(1,1)^{\mathrm{T}}, (1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$ Congruences: $\{(x \equiv_2 1), (x + y \equiv_3 2)\}$

Grid Representation: Equalities and Lines

We also allow for equalities to hold and directions or lines where the vectors of the grid can take any, possibly non-integral, numerical values.

➜ Equalities can be defined as a “modulo 0” congruence relation.

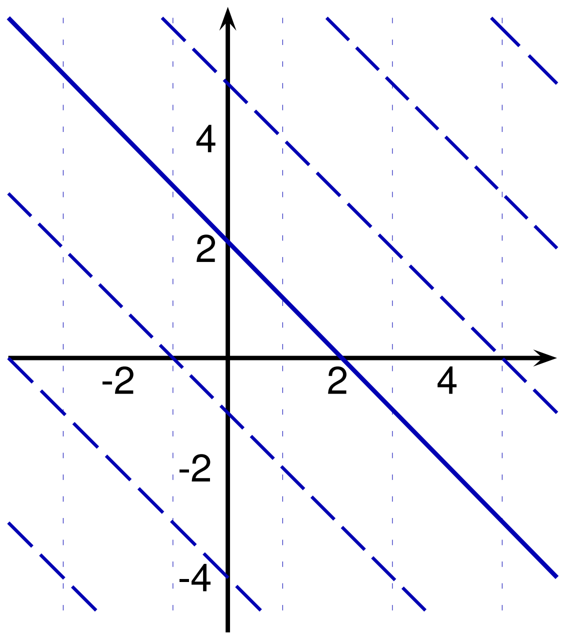
For instance, $(x + y \equiv_0 5)$ denotes the equality $(x + y = 5)$.

➜ A separate set of lines defines a vector space, that is, the directions where the values are unrestricted. Thus a generator system for a grid is a pair of sets of vectors, (L, P)
L is the set of generating lines and
P is the set of generating points.

For instance, $(\{(1,1)^{\mathrm{T}}\}, \{(0,0)^{\mathrm{T}}, (1,0)^{\mathrm{T}}\})$ generates the set of lines that satisfy the congruence $x - y \equiv_1 0$.

Grid Represention: Lines and Vector Spaces

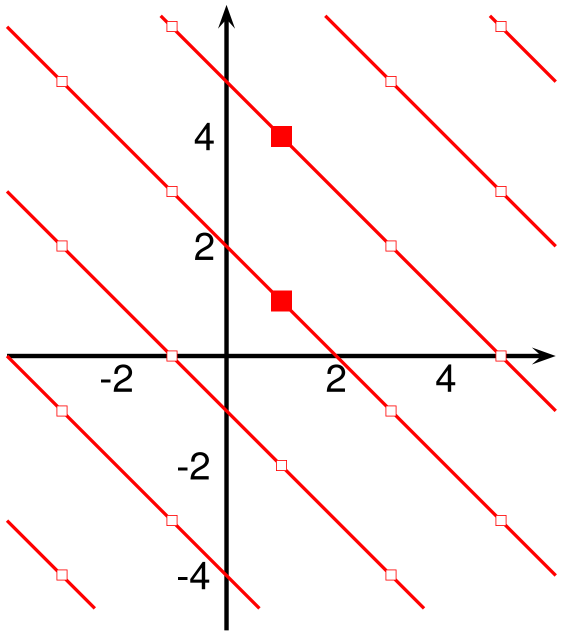
Consider the grid that is described by just the singleton set of congruence relations: $\{(x + y \equiv_3 2)\}$



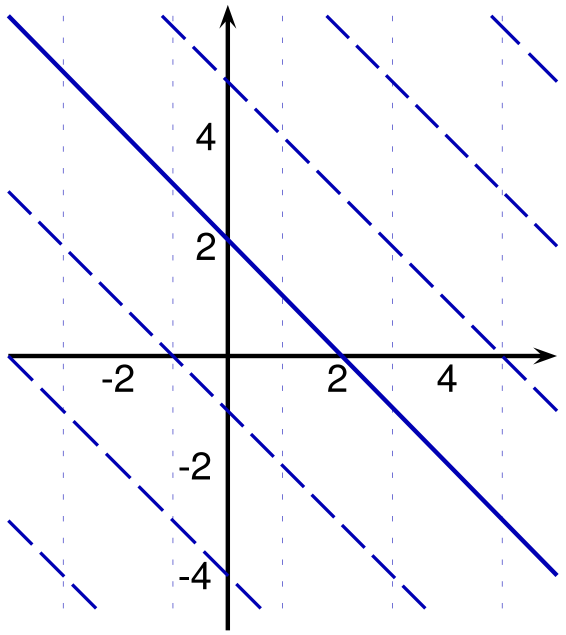
Congruences: $\{(x + y \equiv_3 2)\}$

Grid Represention: Lines and Vector Spaces

Consider the grid that is described by just the singleton set of congruence relations: $\{(x+y \equiv_3 2)\}$



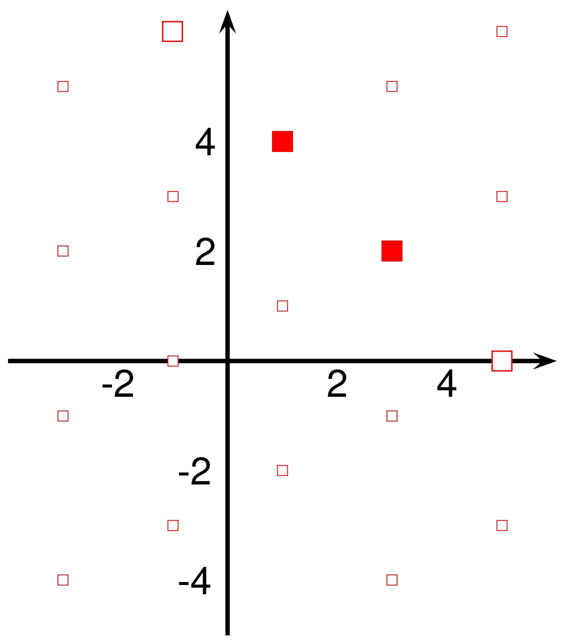
Generators: $\left(\{(-1,1)^{\mathrm{T}}\},\{(1,1)^{\mathrm{T}},(1,4)^{\mathrm{T}}\}\right)$



Congruences: $\{(x+y \equiv_3 2)\}$

Grid Representention: Equalities

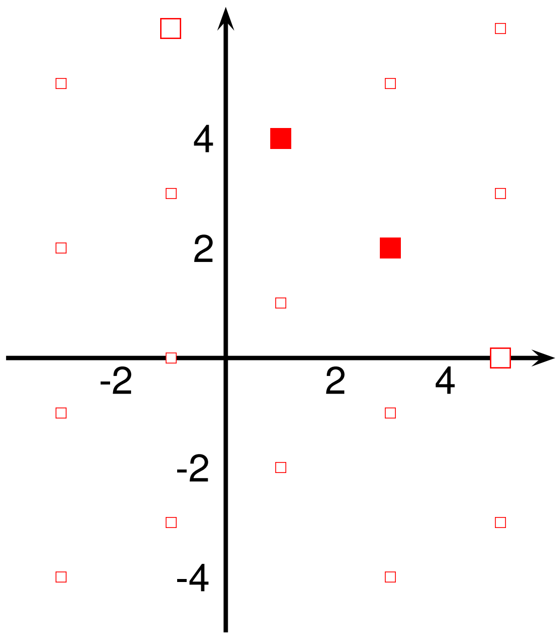
Consider the grid that is generated by just the points $\{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$:



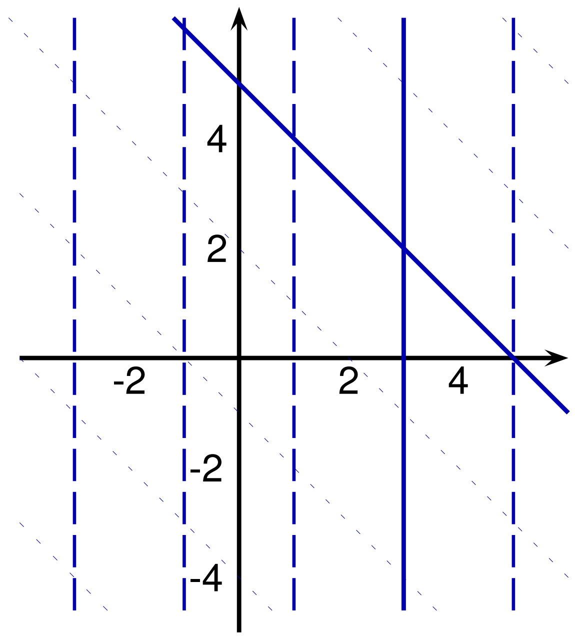
Generators: $\{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$

Grid Represention: Equalities

Consider the grid that is generated by just the points $\{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$:



Generators: $\{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}$



Congruences: $\{(x \equiv_2 1), (x + y = 5)\}$

Generating and Describing a Grid

Suppose $\mathcal{C}$ is a finite set of congruence relations in $\mathbb{Q}^n$.
Then the grid $\mathcal{G}$ described by $\mathcal{C}$ is the set

$$\mathcal{G} = \mathrm{gcon}(\mathcal{C}) := \left\{ \vec{v} \in \mathbb{R}^n \;\middle|\; \forall\big(\langle \vec{a}, \vec{x}\rangle \equiv_k b\big) \in \mathcal{C} : \langle \vec{a}, \vec{v}\rangle \equiv_k b \right\}.$$

Suppose $L, P \subseteq \mathbb{Q}^n$ are finite sets such that $\vec{0} \notin L$.
Then, if $\# L = \ell$ and $\# P = p$, the grid $\mathcal{G}$ generated by (L, P) is

$$\mathcal{G} = \mathrm{ggen}\big((L, P)\big) := \{\, L\vec{\lambda} + P\vec{\pi} \in \mathbb{R}^n \mid \vec{\lambda} \in \mathbb{R}^\ell, \vec{\pi} \in \mathbb{Z}^p, \textstyle\sum_{i=1}^{p} \pi_i = 1 \,\}.$$

Example: Describing a Grid

Consider the set $\mathcal{C}$ of congruence relations,

$$\{(x \equiv_2 1), (x + y \equiv_3 2)\}.$$

Then the grid $\mathcal{G}$ described by $\mathcal{C}$ contains every point $\vec{v} = (v_1, v_2) \in \mathbb{R}^n$ that satisfies

$$\begin{aligned} v_1 &\equiv_2 1, \\ v_1 + v_2 &\equiv_3 2. \end{aligned}$$

In particular, $(1,1)^{\mathrm{T}}, (1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}} \in \mathcal{G}$.

Example: Generating a Grid

Consider the generating points:

$$P = \{(1,1)^{\mathrm{T}}, (1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\},$$

then $P\vec{\pi}$ where $\vec{\pi} \in \mathbb{Z}^3$ and $\sum_{i=1}^{p} \pi_i = 1$ generates all the points in the grid. For instance

$$(1,7)^{\mathrm{T}} = -(1,1)^{\mathrm{T}} + 2 * (1,4)^{\mathrm{T}}, \qquad \vec{\pi} = (-1,2,0)^{\mathrm{T}};$$
$$(3,5)^{\mathrm{T}} = -(1,1)^{\mathrm{T}} + (1,4)^{\mathrm{T}} + (3,2)^{\mathrm{T}}, \qquad \vec{\pi} = (-1,1,1)^{\mathrm{T}}.$$

A Double Description for Grids

Suppose

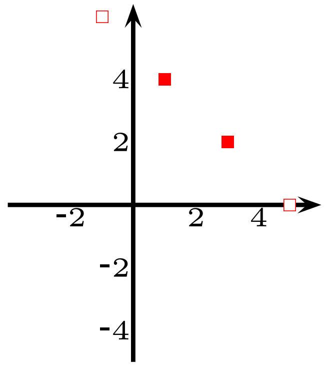
$$\mathcal{G} = \mathrm{gcon}(\mathcal{C}) = \mathrm{ggen}\bigl((L, P)\bigr).$$

Then we say that $\bigl(\mathcal{C}, (L, P)\bigr)$ is a Double Description (DD) pair for $\mathcal{G}$ and we write

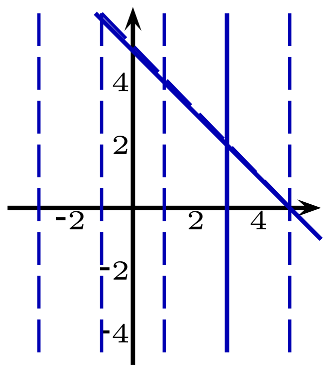
$$\mathcal{G} \equiv \bigl(\mathcal{C}, (L, P)\bigr).$$

Note that it can be shown that a set of points can be described by a congruence system iff there is a generator system that can generate the same set.

Grid DD pairs: Examples

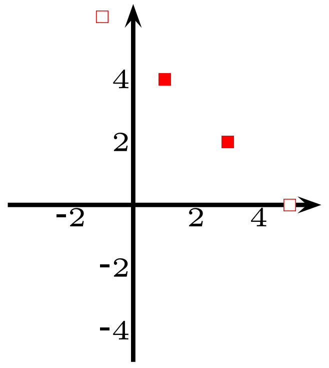


$$(L, P) = \left(\varnothing, \{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}\right)$$

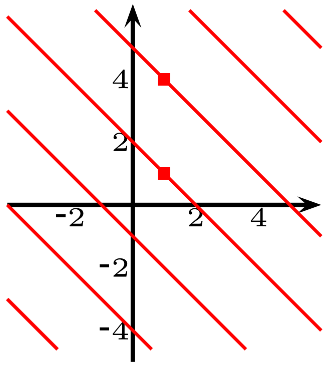


$$\mathcal{C} = \{(x + y = 5), (x \equiv_2 1)\}$$
$$\mathcal{G} \equiv \left(\mathcal{C}, (L, P)\right)$$

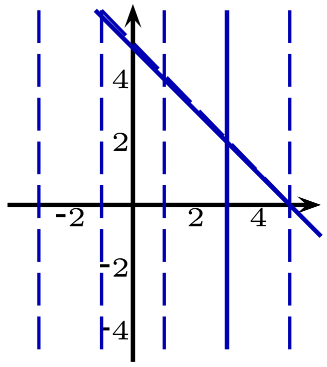
GRID DD PAIRS: EXAMPLES



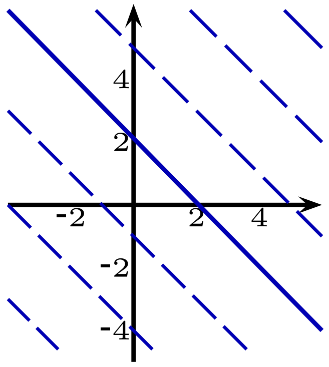
$$(L, P) = \left(\varnothing, \{(1,4)^{\mathrm{T}}, (3,2)^{\mathrm{T}}\}\right)$$



$$(L', P') = \left(\{(-1,1)^{\mathrm{T}}\}, \{(1,1)^{\mathrm{T}}, (1,4)^{\mathrm{T}}\}\right)$$



$$\mathcal{C} = \{(x + y = 5), (x \equiv_2 1)\}$$
$$\mathcal{G} \equiv (\mathcal{C}, (L, P))$$



$$\mathcal{C}' = \{(x + y \equiv_3 2)\}$$
$$\mathcal{G}' \equiv (\mathcal{C}', (L', P'))$$

Operations on Grids: Comparison

We need several grid operations including the key lattice operations of comparison, join and intersection.

Comparison:
Given any two non-empty grids $\mathcal{G} = \operatorname{ggen}\bigl((L, P)\bigr)$ and $\mathcal{G}' = \operatorname{gcon}(\mathcal{C}')$ in $\mathbb{G}_n$, then $\mathcal{G} \subseteq \mathcal{G}'$ if and only if:

$$\forall\bigl(\langle \vec{a}, \vec{x}\rangle \equiv_k b\bigr) \in \mathcal{C}' : \forall \vec{p} \in P : \langle \vec{a}, \vec{p}\rangle \equiv_k b,$$

$$\forall\bigl(\langle \vec{a}, \vec{x}\rangle \equiv_k b\bigr) \in \mathcal{C}' : \forall \vec{\ell} \in L : \langle \vec{a}, \vec{l}\rangle \equiv_k 0.$$

Operations on Grids: Join

Grid-join:

Given any two grids $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{G}_n$, then the grid-join of $\mathcal{G}_1$ and $\mathcal{G}_2$ is the smallest grid containing both $\mathcal{G}_1$ and $\mathcal{G}_2$.

Suppose $\mathcal{G}_i = \mathrm{ggen}\big((L_i, P_i)\big)$, for $i = 1$, 2. Then

$$\mathcal{G}_1 \oplus \mathcal{G}_2 := \mathrm{ggen}\big((L_1 \cup L_2, P_1 \cup P_2)\big)$$

is the grid-join of $\mathcal{G}_1$ and $\mathcal{G}_2$.

Operations on Grids: Intersection

Grid-intersection:

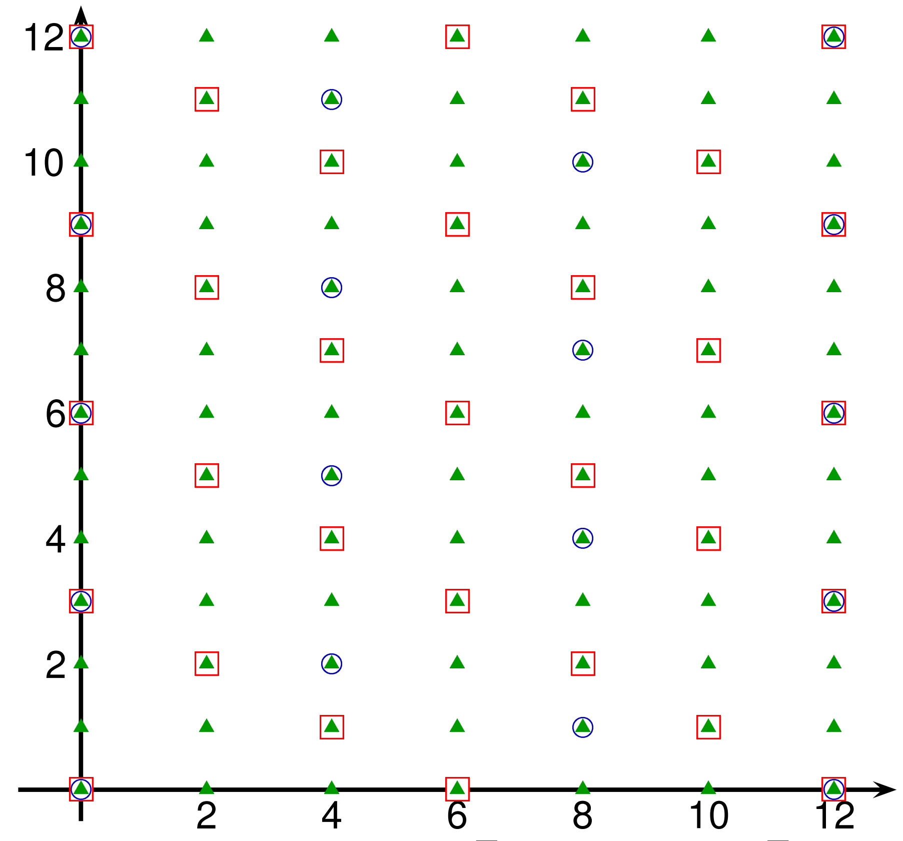
Given a pair of grids $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{G}_n$, the grid-intersection of $\mathcal{G}_1$ and $\mathcal{G}_2$ is the largest grid whose points lie in both $\mathcal{G}_1$ and $\mathcal{G}_2$.

Suppose $\mathcal{G}_i = \mathrm{gcon}(\mathcal{C}_i)$, for $i = 1$, 2. Then

$$\mathcal{G}_1 \cap \mathcal{G}_2 := \mathrm{gcon}(\mathcal{C}_1 \cup \mathcal{C}_2)$$

is the grid-intersection of $\mathcal{G}_1$ and $\mathcal{G}_2$.

Grid Join and Intersection: Example

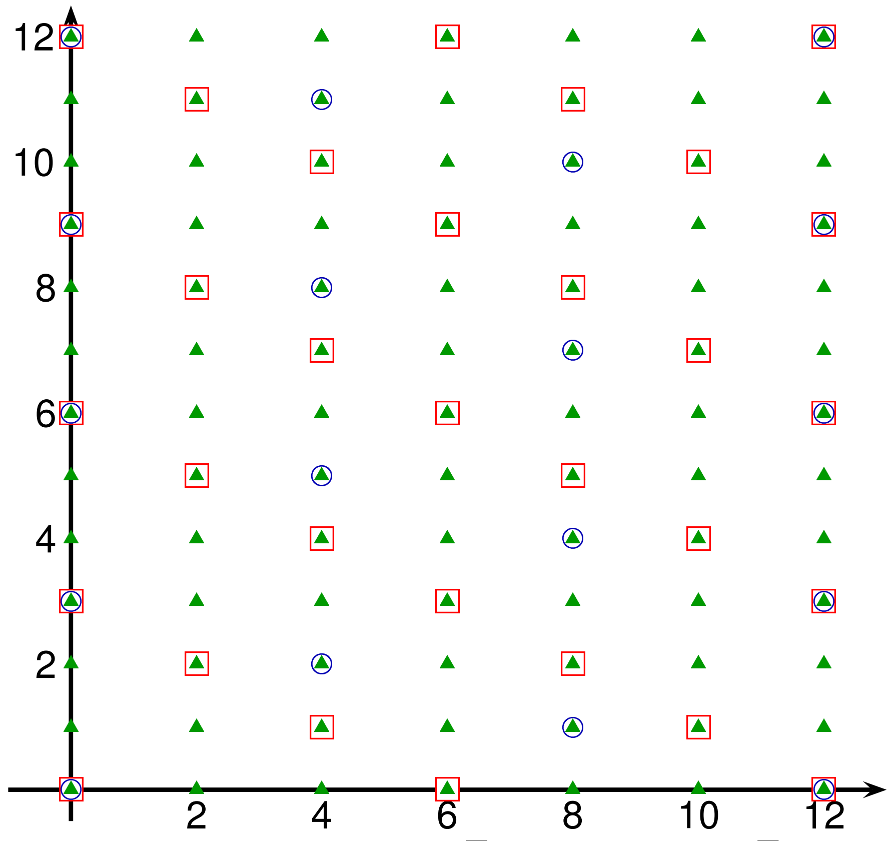


□ $\mathcal{G} = \mathrm{ggen}\big((\varnothing, \{(2,2)^{\mathrm{T}}, (0,3), (0,0)\}^{\mathrm{T}})\big)$

○ $\mathcal{G}' = \mathrm{ggen}\big((\varnothing, \{(4,2)^{\mathrm{T}}, (0,3)^{\mathrm{T}}, (0,0)^{\mathrm{T}}\})\big)$

▲ $\mathcal{G} \oplus \mathcal{G}' = \mathrm{ggen}\big((\varnothing, \{(2,0)^{\mathrm{T}}, (0,1)^{\mathrm{T}}\})\big)$.

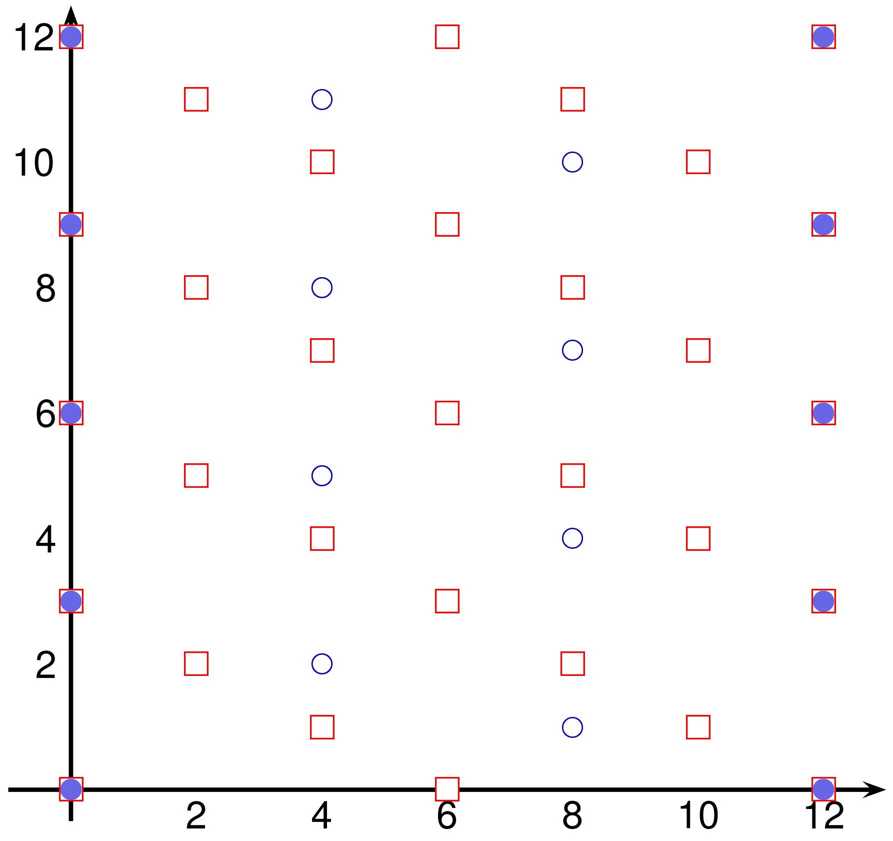
Grid Join and Intersection: Example



□ $\mathcal{G} = \mathrm{ggen}\big((\varnothing, \{(2,2)^{\mathrm{T}}, (0,3), (0,0)\}^{\mathrm{T}})\big)$

○ $\mathcal{G}' = \mathrm{ggen}\big((\varnothing, \{(4,2)^{\mathrm{T}}, (0,3)^{\mathrm{T}}, (0,0)^{\mathrm{T}}\})\big)$

▲ $\mathcal{G} \oplus \mathcal{G}' = \mathrm{ggen}\big((\varnothing, \{(2,0)^{\mathrm{T}}, (0,1)^{\mathrm{T}}\})\big)$.



□ $\mathcal{G} = \mathrm{gcon}\big(\{(x \equiv_2 0), (-x + y \equiv_3 0)\}\big)$

○ $\mathcal{G}' = \mathrm{gcon}\big(\{(x \equiv_4 0), (-x + 2y \equiv_6 0)\}\big)$

● $\mathcal{G} \cap \mathcal{G}' = \mathrm{gcon}\big(\{(x \equiv_{12} 0), (y \equiv_3 0)\}\big)$.

Grid Representations: Reduction and Conversion

It can be seen from these operations that we need algorithms:

- ➜ For reducing the size of the representations, since any grid in $\mathbb{G}_n$ can be represented by:
  - ➜ a reduced set of congruences of cardinality of at most n;
  - ➜ a reduced set of lines and points of cardinality of at most $n + 1$.
- ➜ For converting from one representation to the other.

Grid: Reduced Form for Congruence Systems

The max number of congruences needed to describe a grid is n.

Examples: Suppose

$$\begin{aligned} \mathcal{C}_1 &= \bigl\{(x \equiv_2 0), (x \equiv_2 1)\bigr\}; & \mathcal{C}'_1 &= \bigl\{(x \equiv_3 0), (x \equiv_3 1)\bigr\}; \\ \mathcal{C}_2 &= \bigl\{(x \equiv_1 0), (x \equiv_2 1)\bigr\}; & \mathcal{C}'_2 &= \bigl\{(x \equiv_2 1)\bigr\}; \\ \mathcal{C}_3 &= \bigl\{(x \equiv_3 2), (x \equiv_2 1)\bigr\}; & \mathcal{C}'_3 &= \bigl\{(x \equiv_6 5)\bigr\}. \end{aligned}$$

Both $\mathcal{C}_1$ and $\mathcal{C}'_1$ are inconsistent and both describe the empty grid.

Congruence $(x \equiv_1 0)$ is redundant in $\mathcal{C}_2$ so that $\mathrm{gcon}(\mathcal{C}_2) = \mathrm{gcon}(\mathcal{C}'_2)$.

Although neither $(x \equiv_3 2)$ nor $(x \equiv_2 1)$ are redundant,
$\mathrm{gcon}(\mathcal{C}_3) = \mathrm{gcon}(\mathcal{C}'_3)$ and $\#\,\mathcal{C}'_3 = 1 < \#\,\mathcal{C}_3 = 2$.

Grid: Reduced Form for Generator Systems

Similarly, the max number of lines and points needed to generate a grid is $n+1$.

Examples:

$$(L_1, P_1) = \Big(\{(1)\},\ \varnothing\Big); \quad (L_1', P_1) = \Big(\varnothing,\ \varnothing\Big).$$
$$\mathrm{ggen}\big((L_1, P_1)\big) = \mathrm{ggen}\big((L_1', P_1)\big) = \varnothing$$

Only (L_1', P_1) is in reduced form.

$$(L_2, P_2) = \Big(\varnothing,\ \big\{(3), (5), (8)\big\}\Big); \quad (L_2, P_2') = \Big(\varnothing,\ \big\{(3), (4)\big\}\Big)$$
$$(L_2, P_2'') = \Big(\varnothing,\ \big\{(0), (1)\big\}\Big)$$
$$\mathrm{ggen}\big((L_2, P_2)\big) = \mathrm{ggen}\big((L_2, P_2')\big) = \mathrm{ggen}\big((L_2, P_2'')\big)$$

Although no point in P_2 is redundant, (L_2, P_2) is not reduced.

Reduced form: generator systems. (Cont.)

$$(L_3, P_3) = \Bigl(\bigl\{(0,1)^{\mathrm{T}}, (1,1)^{\mathrm{T}}, (1,2)^{\mathrm{T}}\bigr\},\ \bigl\{(0,0)^{\mathrm{T}}, (1,0)^{\mathrm{T}}\bigr\}\Bigr);$$

$$(L_3', P_3) = \Bigl(\bigl\{0,1)^{\mathrm{T}}, (1,1)^{\mathrm{T}}\bigr\},\ \bigl\{(0,0)^{\mathrm{T}}, (1,0)^{\mathrm{T}}\bigr\}\Bigr);$$

$$(L_3', P_3') = \Bigl(\bigl\{0,1)^{\mathrm{T}}, (1,1)^{\mathrm{T}}\bigr\},\ \bigl\{(0,0)^{\mathrm{T}}\bigr\}\Bigr).$$

The line $(1,2)^{\mathrm{T}}$ is redundant in L_3.

But $(1,0)^{\mathrm{T}}$ is a redundant point in (L_3', P_3) since, letting:

$$\lambda_1 = 1, \lambda_2 = -1, \pi_1 = 1,$$

$$(1,0)^{\mathrm{T}} = \lambda_1 (1,1)^{\mathrm{T}} + \lambda_2 (0,1)^{\mathrm{T}} + \pi_1 (0,0)^{\mathrm{T}},$$

and $\mathrm{ggen}\bigl((L_3, P_3)\bigr) = \mathrm{ggen}\bigl((L_3', P_3)\bigr) = \mathrm{ggen}\bigl((L_3', P_3')\bigr)$.

GRIDS: RELATED WORK

① Text books on linear and integer programming and optimizations describe a parameter representation (similar to the generator representation) for full dimensional, integral grids.
  ➜ Schriver (1986) and
  ➜ Nemhauser and Wolsey (1988);

② A domain of full dimensional, integral grids for $\mathcal{Z}$-polyhedra
  ➜ Ancourt (1991),
  ➜ Quinton, Rajopadhye and Risset (1996), and
  ➜ Nookala and Risset (2000);

③ Congruence and parameter representations for integral and rational grids with algorithms for conversion and reduction, although these are mainly for integral pointed grids
  ➜ Granger (1989, 1991 and 1997).

Summary of Future Plans

New releases for the Parma Polyhedra Library are planned for July (0.6), December (0.7) and in 2005 and beyond (0.8, 0.9, . . .):

- ➜ Many new domains including intervals and bounding boxes, bounded differences and quotients, grids, and octagons.
- ➜ New domain constructions including powersets, ask-and-tell, and generalizations of $\mathbb{Z}$-polyhedra to $\mathbb{G}$-$\star$.
- ➜ New language interfaces including OCaml and Java.
- ➜ New operators such as the ones for summarization, and serialization and deserialization.
- ➜ Other new features such as the simplex algorithm, native integers with overflow detection, and cartesian factoring.

For More Information:

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

There you can find:

➜ releases and development snapshots;

➜ documentation and installation instructions;

➜ mailing lists for users and developers;

➜ papers and technical reports;

➜ slides of our presentations;

➜ an extensive bibliography (in BibTeX format) on the subject;

➜ links to relevant sources of information.