Convex Polyhedra for the Analysis and Verification of Hardware and Software Systems: the “Parma Polyhedra Library”

Roberto BAGNARA, Patricia M. HILL, Enea ZAFFANELLA,
Elisa RICCI, Sara BONINI, Andrea PESCETTI,
Angela STAZZONE, Tatiana ZOLO, Elena MAZZI,
Barbara QUARTIERI, Katy DOBSON

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

Plan of the Talk

THE PROBLEM

- Hardware is millions of times more powerful than it was 25 years ago;
- program sizes have exploded in similar proportions;
- large and very large programs (up to tens of millions of lines of code) are and will be in widespread use;
- they need to be designed, developed and maintained over their entire lifespan (up to 20 and more years) at reasonable costs;
- unassisted development and maintenance teams do not stand a chance to follow such an explosion in size and complexity;
- many pieces of software exhibit a number of bugs that is sometimes 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.

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).

FORMAL PROGRAM VERIFICATION METHODS

Purpose

- To mechanically prove that all possible program executions are correct in all specified execution environments...
- ...for some definition of correct:
  - absence of some kinds of run-time errors;
  - adherence to some partial specification...

Several methods

- deductive methods;
- model checking;
- program typing;
- static analysis.

Because of the undecidability of program verification

- all methods are partial or incomplete;
- all resort to some form of approximation.

Abstract Interpretation

- The right framework to work with the concept of sound approximation;
- a theory for approximating sets and set operations as considered in set (or category) theory, including inductive definitions;
- a theory of approximation of the behavior of dynamic discrete systems;
- Computation takes place on a domain of abstract properties: the abstract domain. . .
- . . . using abstract operations which are sound approximations of the concrete operations.
- Correctness follows by design!
- The abstraction (approximation) can be coarse enough to be finitely computable, yet be precise enough to be practically useful.
- Examples: casting out of nines and rule of signs.

Convex Polyhedra and Beyond

What?

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

Restrictions, interesting for efficiency reasons:

- ➜ bounding boxes;
- ➜ systems of bounded differences;
- ➜ octagons.

Generalizations and extensions, interesting for expressivity reasons:

- ➜ not necessarily closed polyhedra (boxes, differences, octagons);
- ➜ grids;
- ➜ trapezoidal congruences;
- ➜ intersections of the above ($\mathbb{Z}$-polyhedra);
- ➜ sets of the above (sets of bounding boxes, sets of polyhedra, sets of grids, sets of $\mathbb{Z}$-polyhedra, . . .).

WHY ARE THESE INTERESTING AND USEFUL?

Solving classical data-flow analysis problems!

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

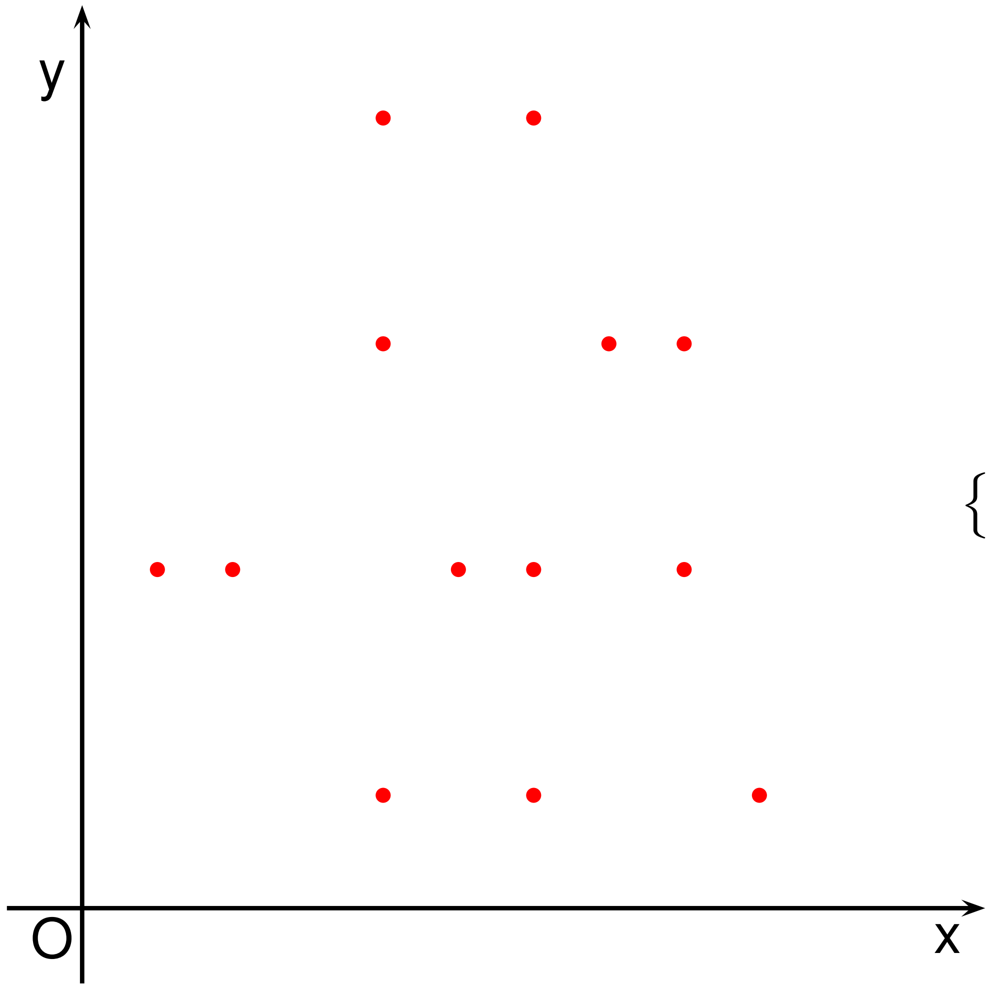
Verification of concurrent and reactive systems!

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

And again: many other applications...

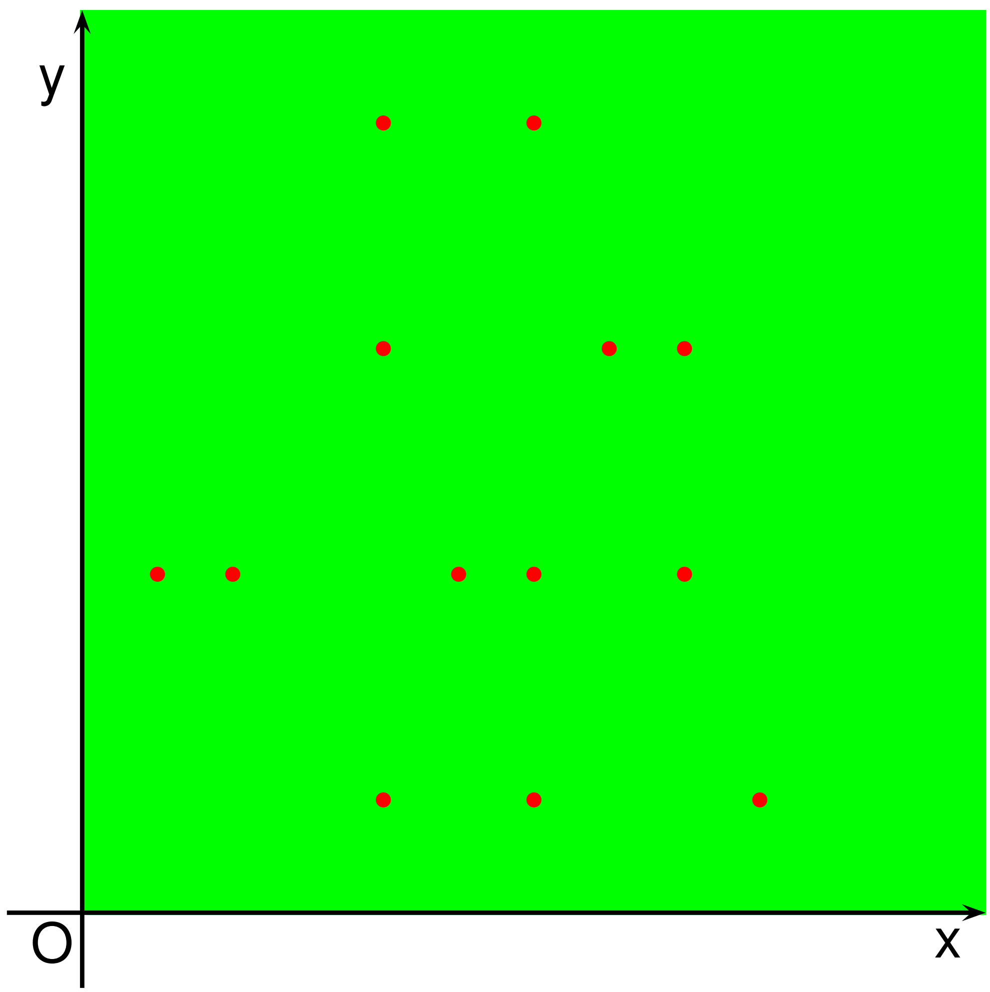
- ➜ inferring argument size relationships in logic programs;
- ➜ termination inference for logic and functional programs.

Numerical Abstractions: No Abstraction



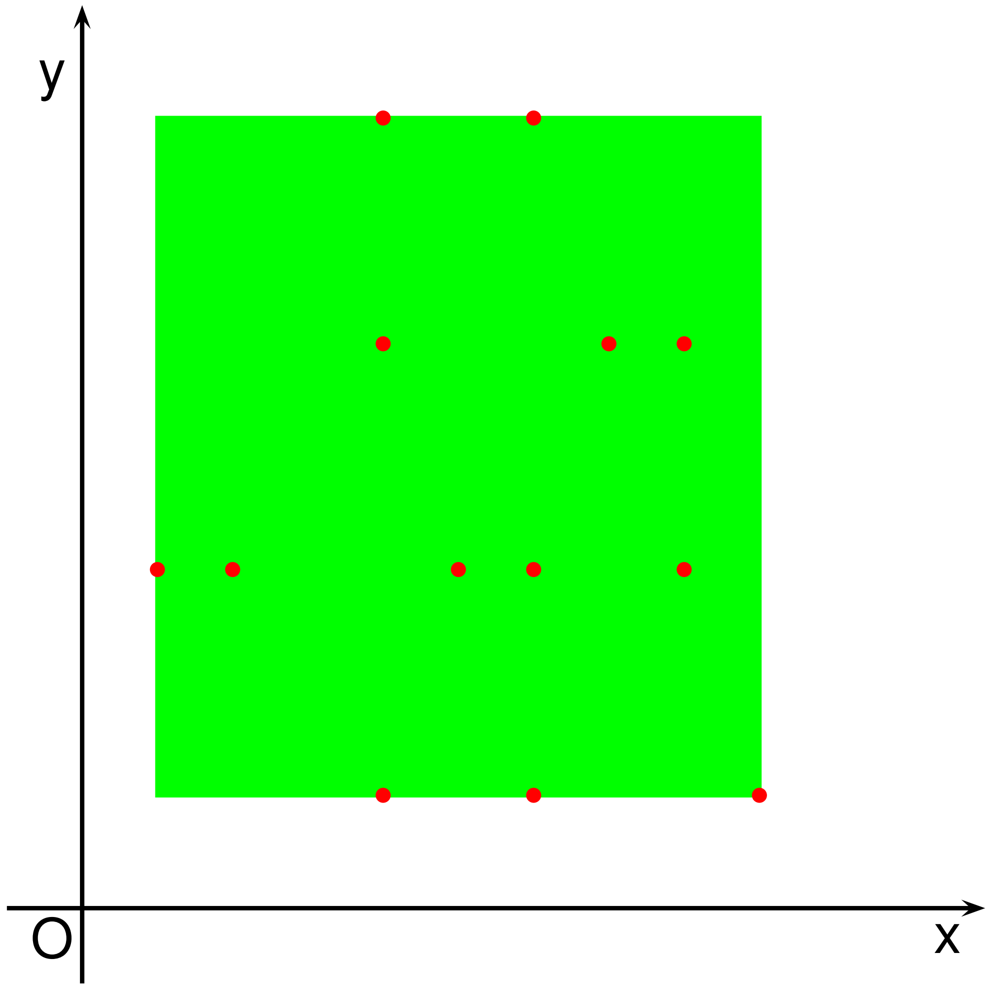
$$\{\ldots, (2, 9), \ldots, (12, 21) \ldots, \}$$

Numerical Abstractions: Signs



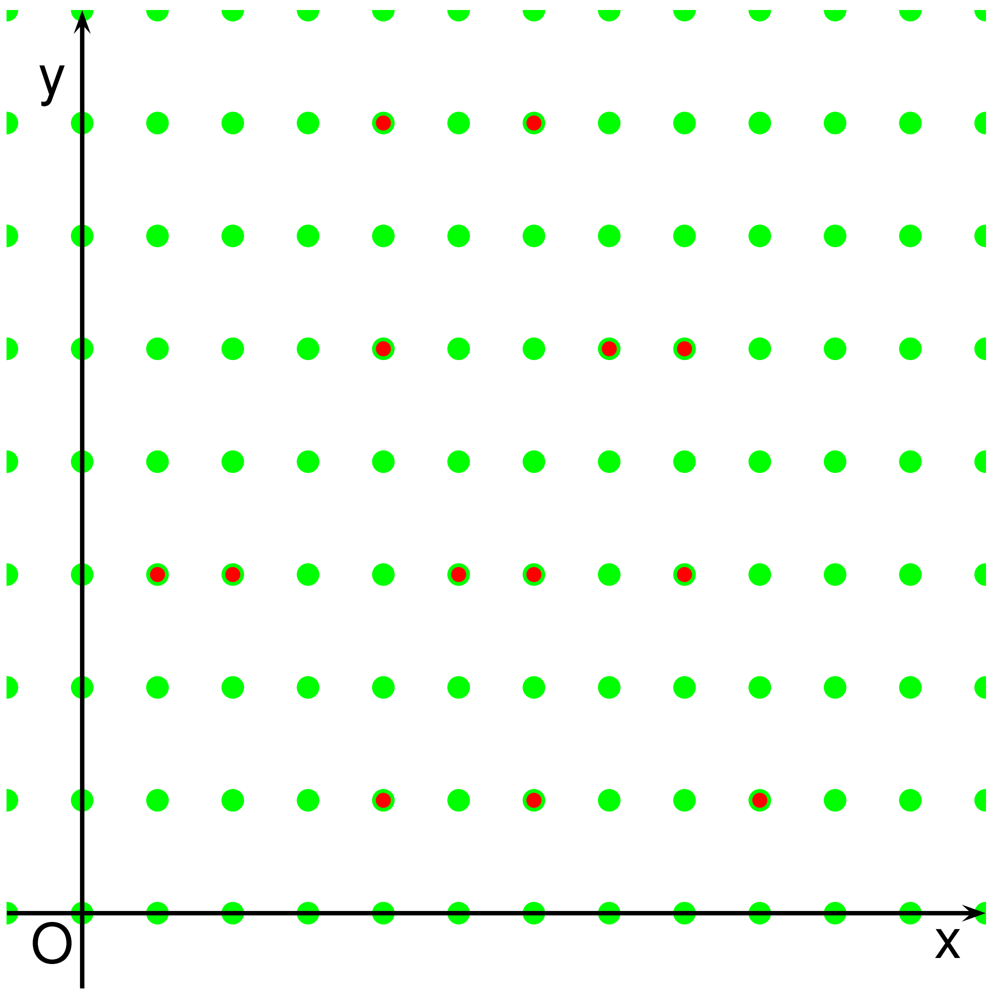
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Numerical Abstractions: Bounding Boxes



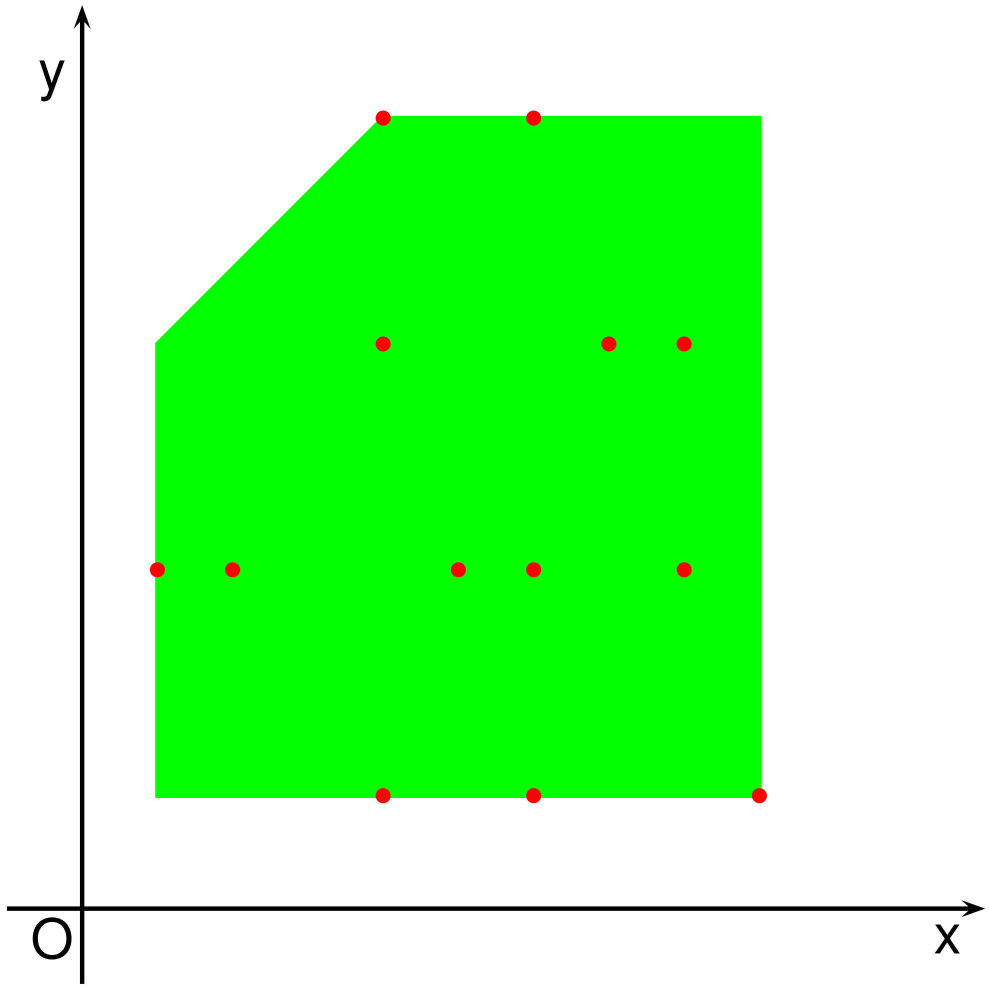
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \end{cases}$$

Numerical Abstractions: Simple Congruences



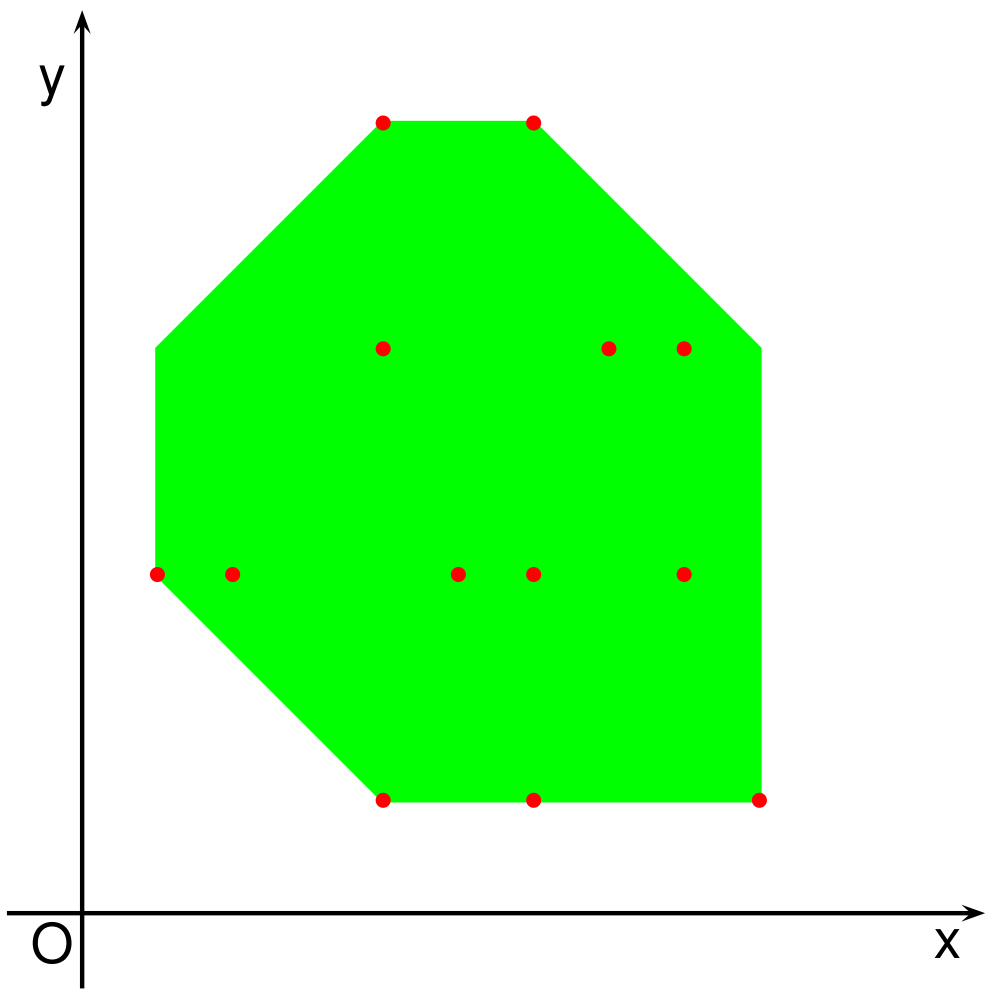
$$\begin{cases} x = 0 \mod 2 \\ y = 0 \mod 3 \end{cases}$$

Numerical Abstractions: Bounded Differences



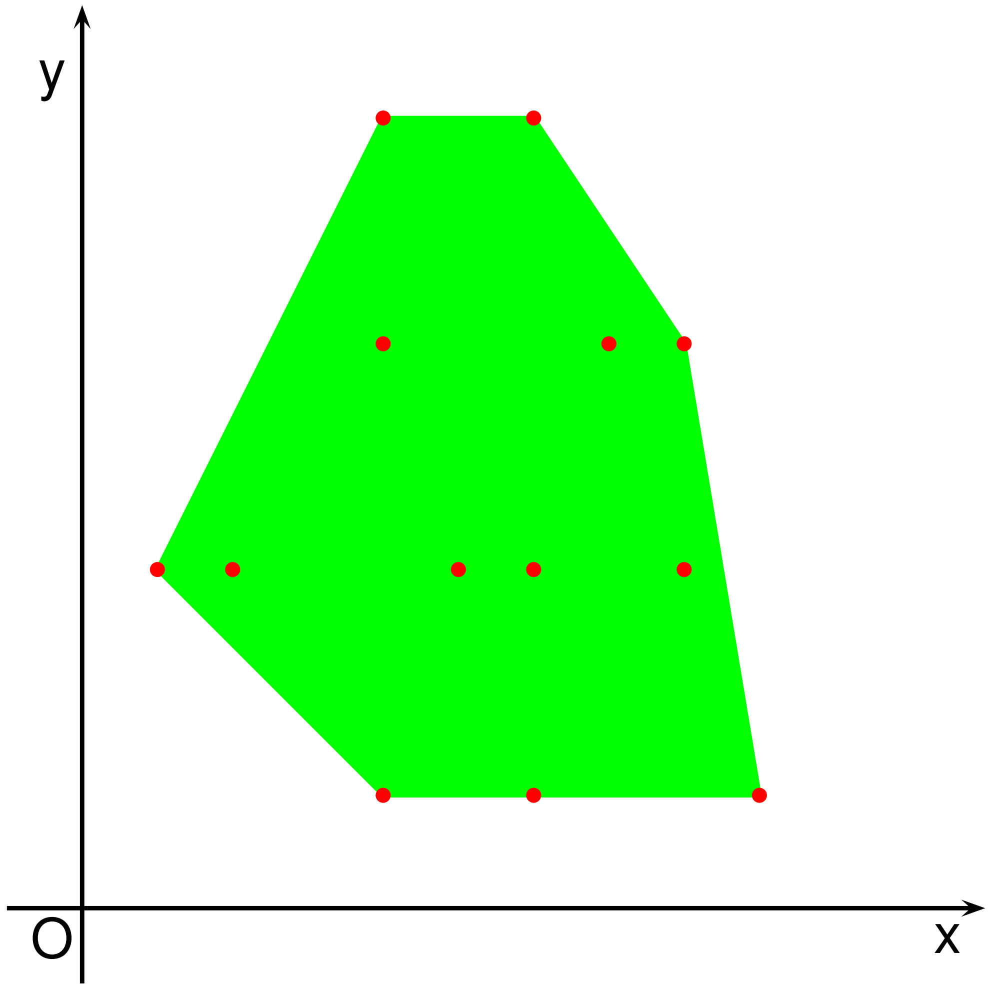
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \end{cases}$$

Numerical Abstractions: Octagons



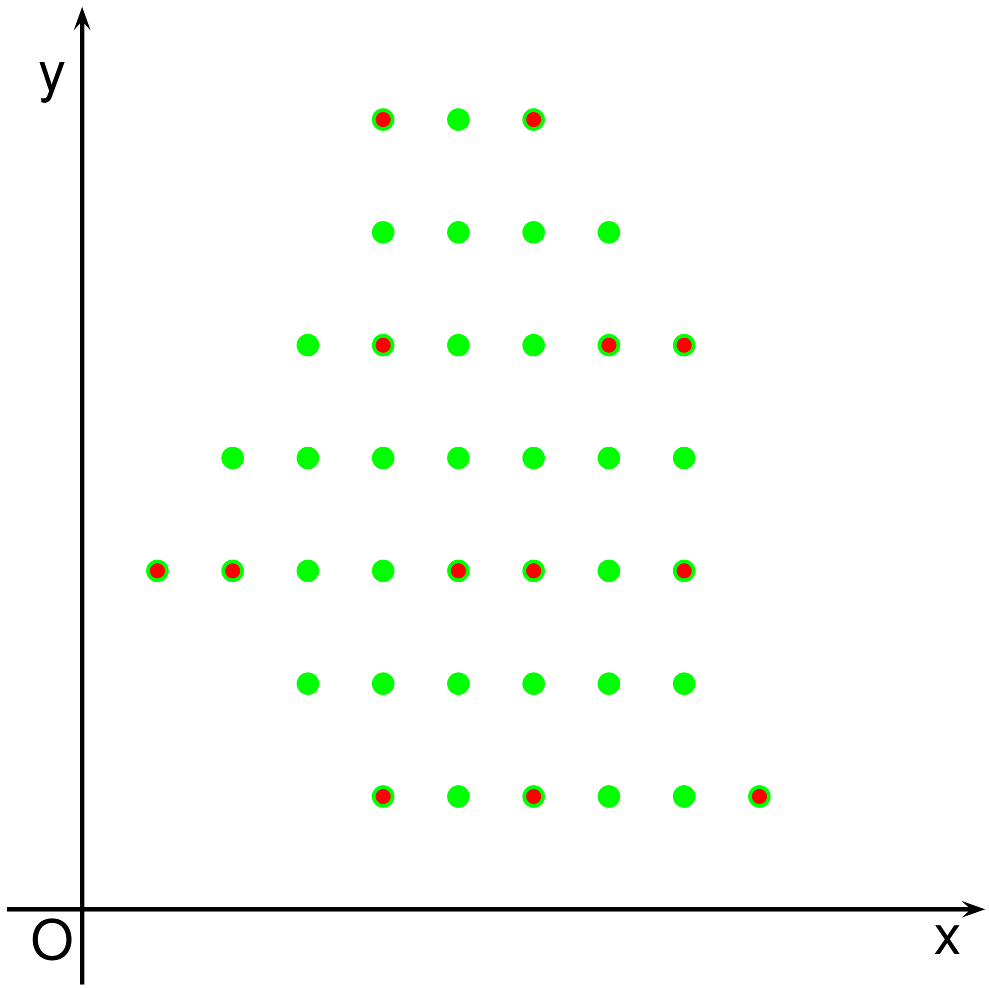
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \\ 11 \leq x + y \leq 33 \end{cases}$$

Numerical Abstractions: Convex Polyhedra



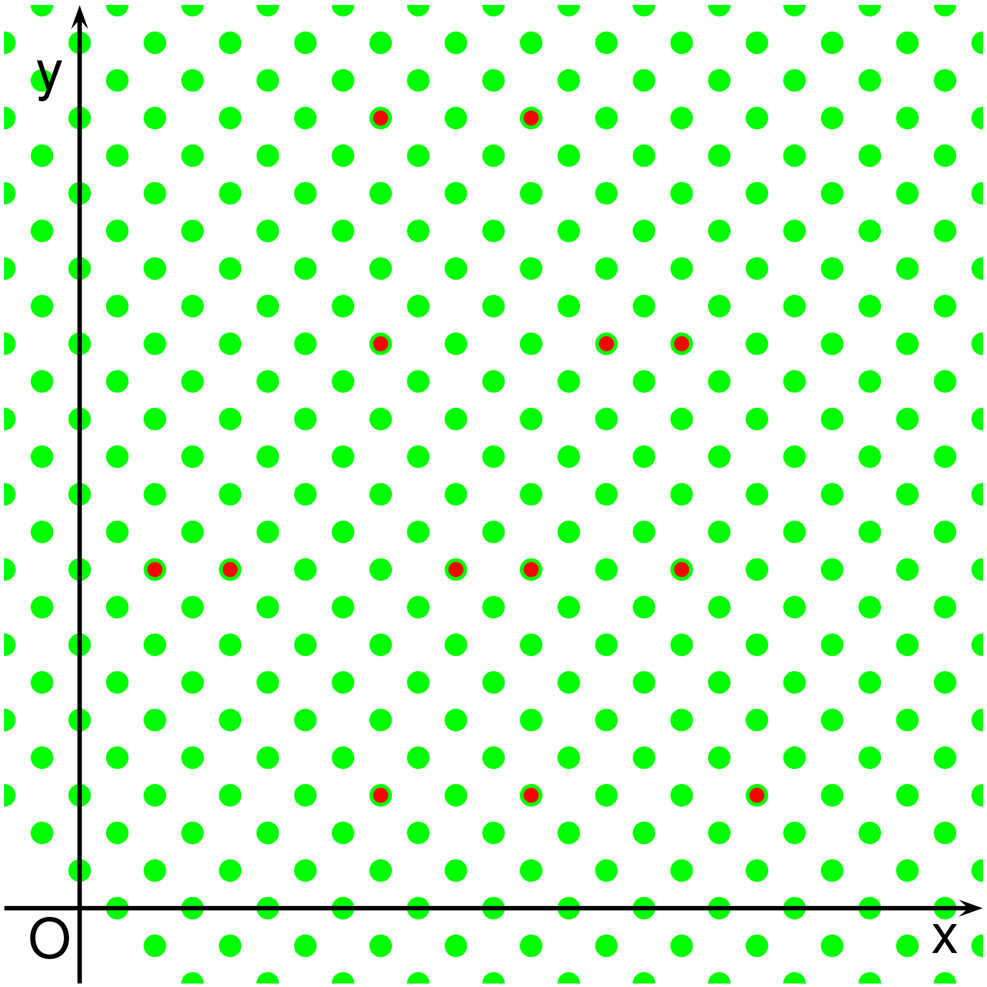
$$
\begin{cases}
6x + y \leq 111 \\
3x + 2y \leq 78 \\
x + y \geq 11 \\
2x - y \geq -5 \\
y \geq 3 \\
y \leq 21
\end{cases}
$$

Numerical Abstractions: $\mathbb{Z}$-Polyhedra (I)



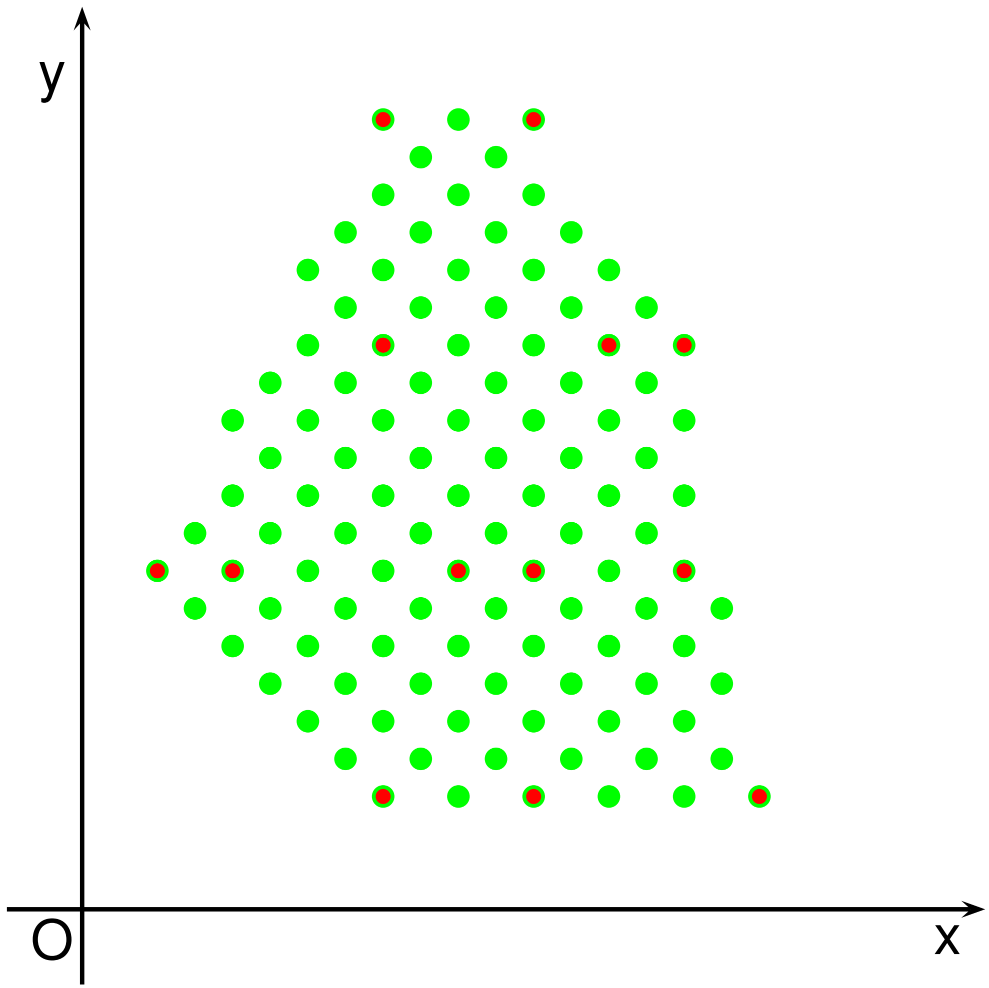
$$\begin{cases} x = 0 \mod 2 \\ y = 0 \mod 3 \\ 6x + y \leq 111 \\ 3x + 2y \leq 78 \\ x + y \geq 11 \\ 2x - y \geq -5 \\ y \geq 3 \\ y \leq 21 \end{cases}$$

Numerical Abstractions: Relational Congruences



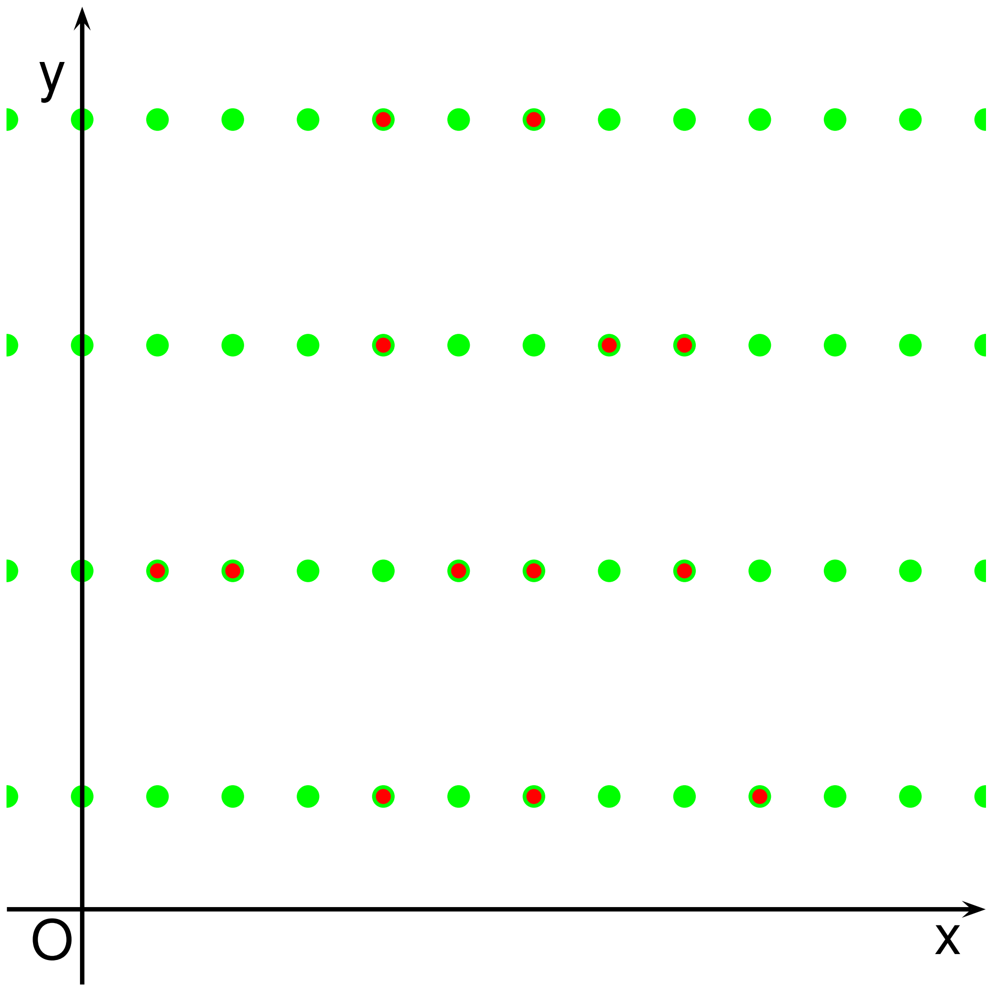
$$\begin{cases} x + y = 1 \mod 2 \end{cases}$$

Numerical Abstractions: $\mathbb{Z}$-Polyhedra (II)



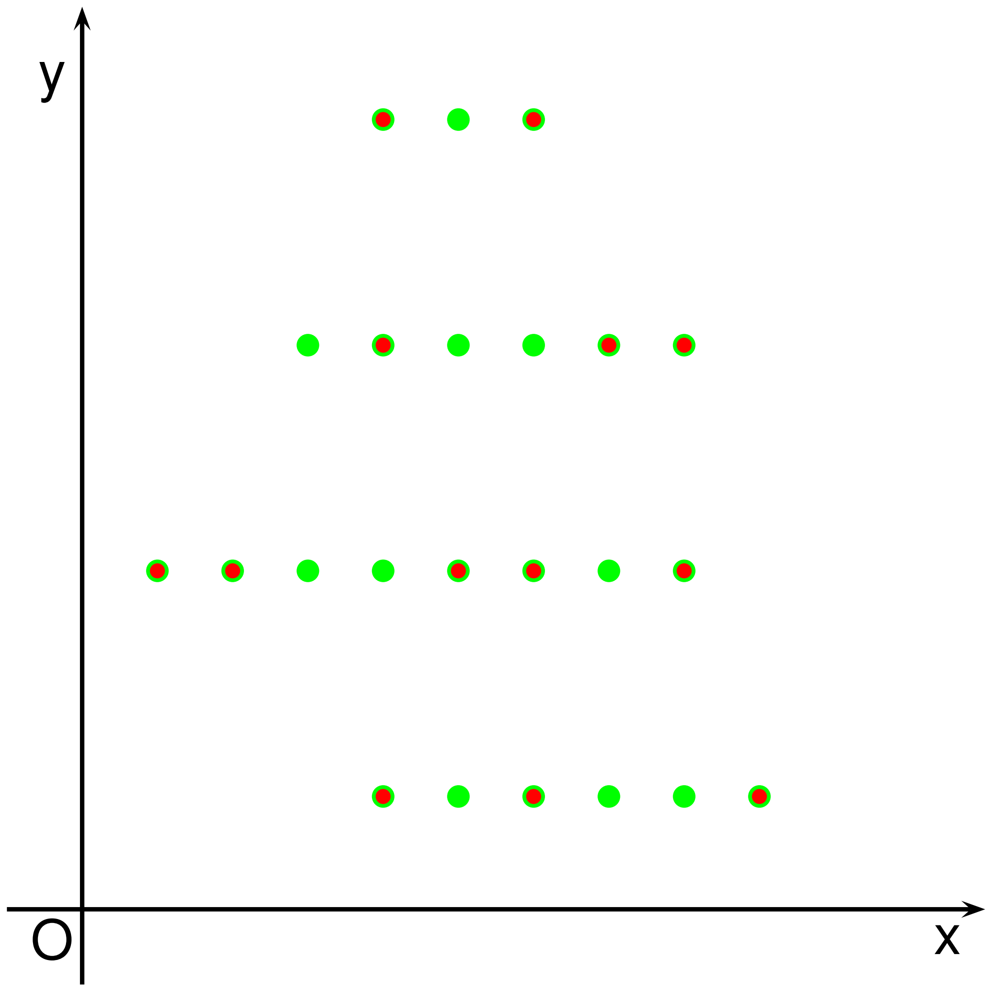
$$\begin{cases} x + y = 1 \mod 2 \\ 6x + y \le 111 \\ 3x + 2y \le 78 \\ x + y \ge 11 \\ 2x - y \ge -5 \\ y \ge 3 \\ y \le 21 \end{cases}$$

Numerical Abstractions: Relational Congruences (II)



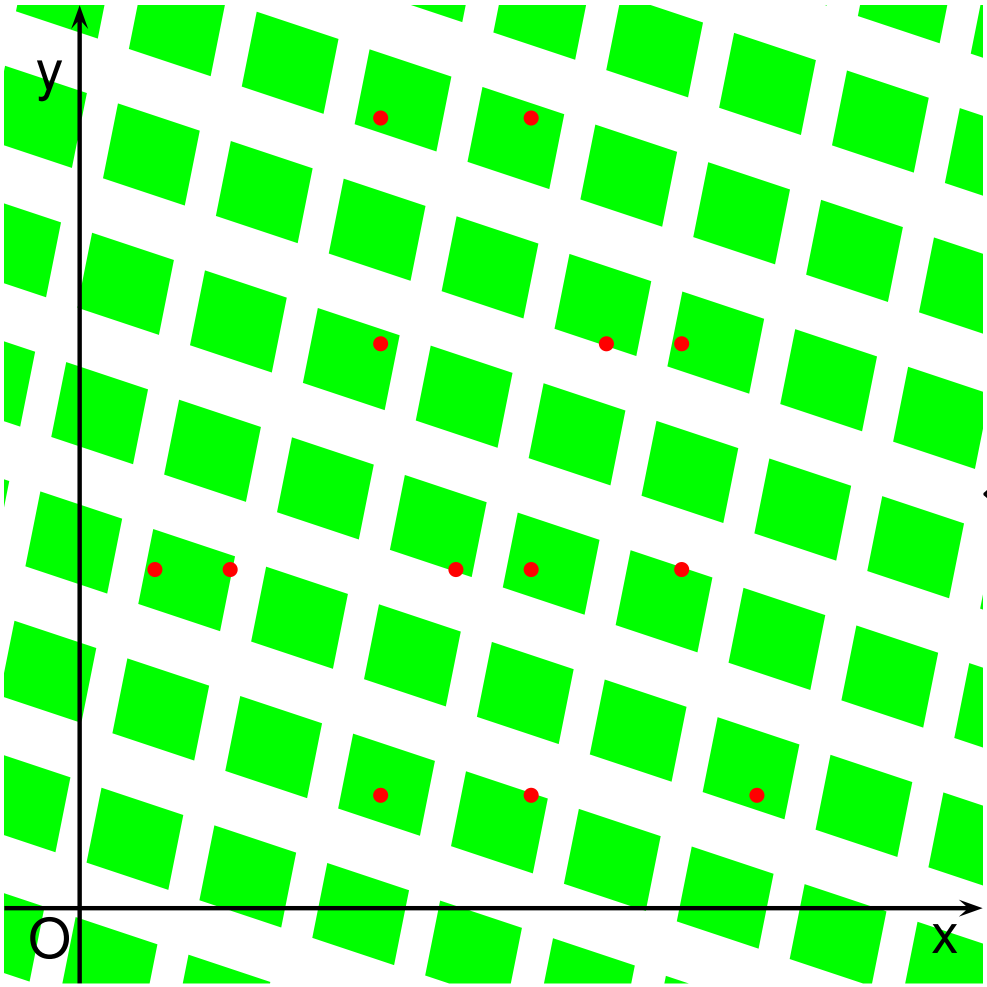
$$\begin{cases} x = 0 & \mod 2 \\ y = 0 & \mod 3 \\ x + y = 1 & \mod 2 \end{cases}$$

Numerical Abstractions: $\mathbb{Z}$-Polyhedra (III)



$$\begin{cases} x = 0 \mod 2 \\ y = 0 \mod 3 \\ x + y = 1 \mod 2 \\ 6x + y \leq 111 \\ 3x + 2y \leq 78 \\ x + y \geq 11 \\ 2x - y \geq -5 \\ y \geq 3 \\ y \leq 21 \end{cases}$$

Numerical Abstractions: Trapezoidal Congruences



$$\begin{cases} x + 3y \in [4, 10] \mod 11 \\ 5x - y \in [0, 11] \mod 16 \end{cases}$$

EXAMPLE: ANALYSIS OF IMPERATIVE PROGRAMS

```
x := 0; y := 0;

while x <= 100 do

  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif

endwhile
```

Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do

  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif

endwhile
```

Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$x = y = 0$

```
  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif

endwhile
```

Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$x = y = 0$

```
  read(b);
  if b then x := x+2
```

$x = 2, y = 0$

```
  else x := x+1; y := y+1;
```

$x = 1, y = 1$

```
  endif

endwhile
```

Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$x = y = 0$

```
  read(b);
  if b then x := x+2
```

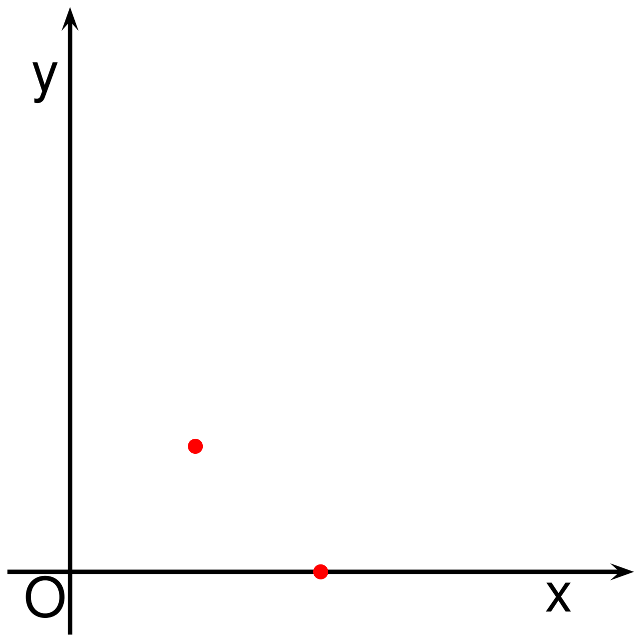
$x = 2, y = 0$

```
  else x := x+1; y := y+1;
```

$x = 1, y = 1$

```
  endif

endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$x = y = 0$

```
  read(b);
  if b then x := x+2
```

$x = 2, y = 0$

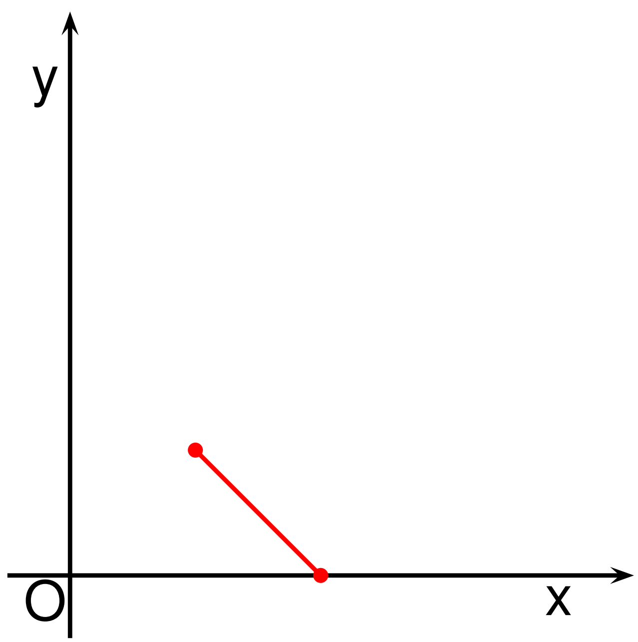
```
  else x := x+1; y := y+1;
```

$x = 1, y = 1$

```
  endif
```

$1 \leq x \leq 2, x + y = 2$

```
endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

?

```
  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif
```

$1 \leq x \leq 2, x + y = 2$

```
endwhile
```

Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

?

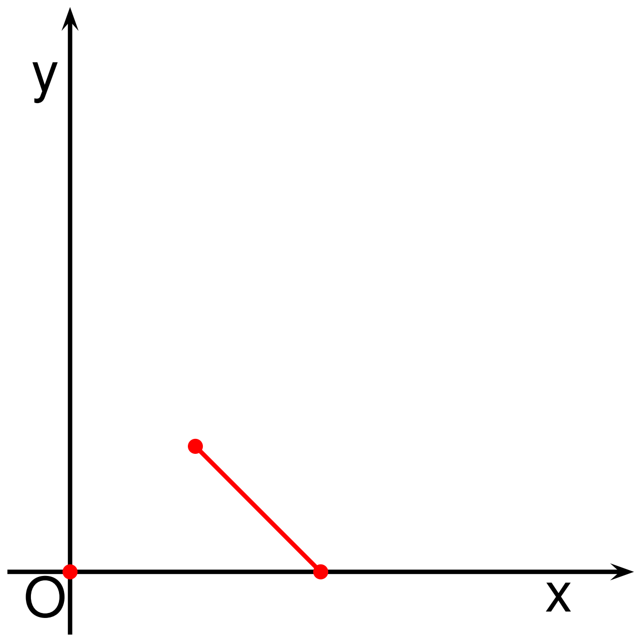
```
  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif
```

$1 \leq x \leq 2, x + y = 2$

```
endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

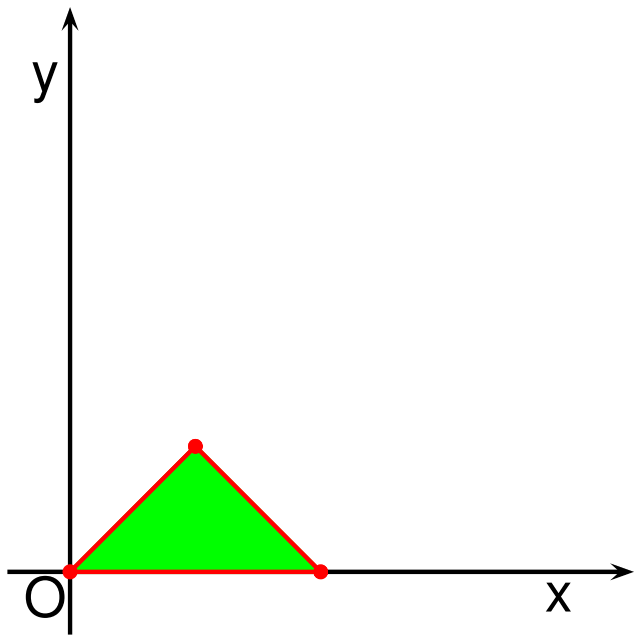
```
  read(b);
  if b then x := x+2

  else x := x+1; y := y+1;

  endif
```

$1 \leq x \leq 2, x + y = 2$

```
endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

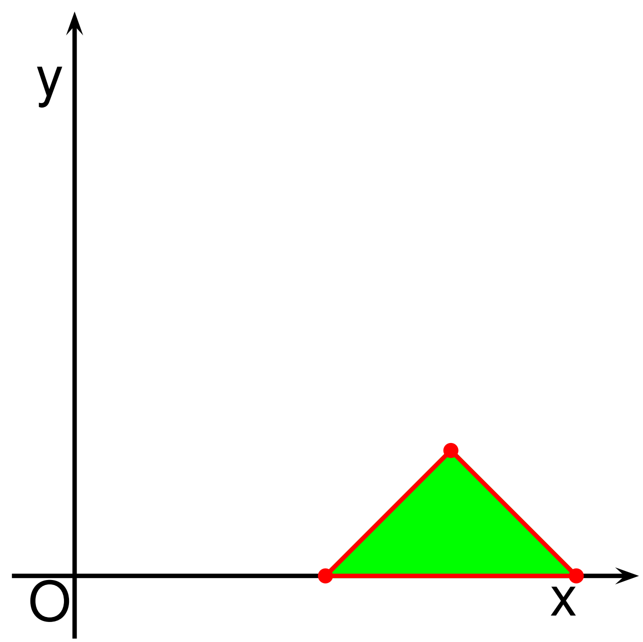
```
  read(b);
  if b then x := x+2
```

$0 \leq y \leq x - 2, x + y \leq 4$

```
  else x := x+1; y := y+1;

  endif

endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

```
  read(b);
  if b then x := x+2
```

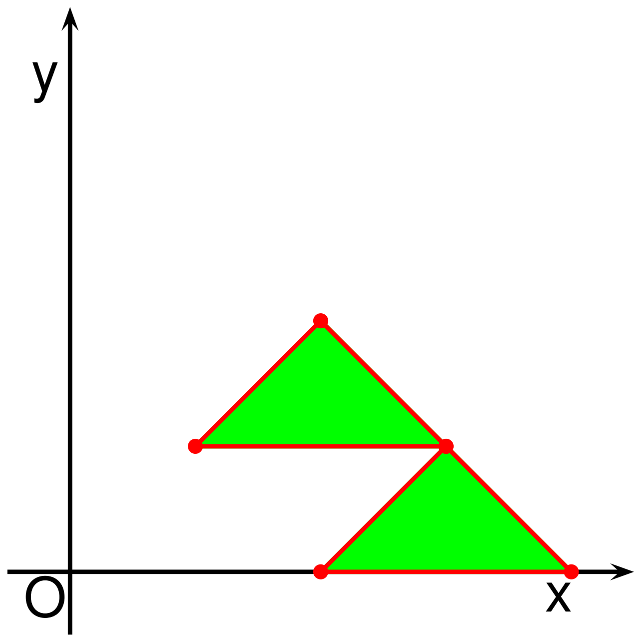
$0 \leq y \leq x - 2, x + y \leq 4$

```
  else x := x+1; y := y+1;
```

$1 \leq y \leq x, x + y \leq 4$

```
  endif

endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

```
  read(b);
  if b then x := x+2
```

$0 \leq y \leq x - 2, x + y \leq 4$

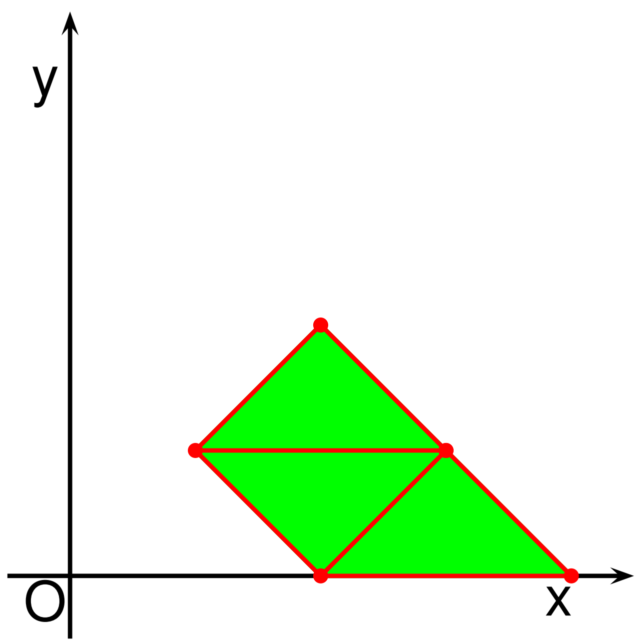
```
  else x := x+1; y := y+1;
```

$1 \leq y \leq x, x + y \leq 4$

```
  endif
```

$0 \leq y \leq x, 2 \leq x + y \leq 4$

```
endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

```
  read(b);
  if b then x := x+2
```

$0 \leq y \leq x - 2, x + y \leq 4$

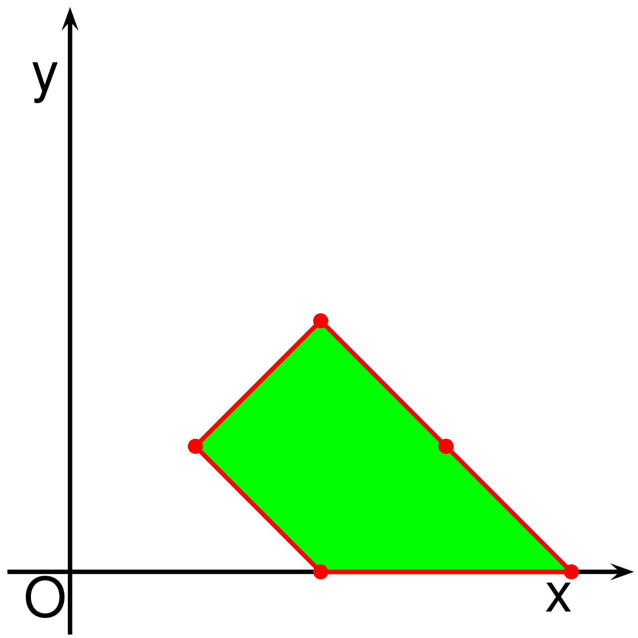
```
  else x := x+1; y := y+1;
```

$1 \leq y \leq x, x + y \leq 4$

```
  endif
```

$0 \leq y \leq x, 2 \leq x + y \leq 4$

```
endwhile
```



Example: Analysis of Imperative Programs

```
x := 0; y := 0;
```

$x = y = 0$

```
while x <= 100 do
```

$0 \leq y \leq x, x + y \leq 2$

```
  read(b);
  if b then x := x+2
```

$0 \leq y \leq x - 2, x + y \leq 4$

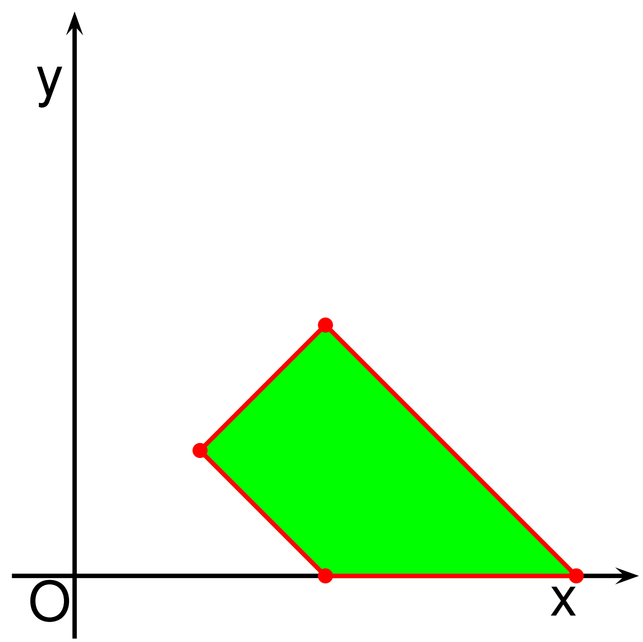
```
  else x := x+1; y := y+1;
```

$1 \leq y \leq x, x + y \leq 4$

```
  endif
```

$0 \leq y \leq x, 2 \leq x + y \leq 4$

```
endwhile
```



Another Example: Validation of Array References

```
heapsort(int n, float t[n])
```

$$n \geq 2$$

```
int l := (n div 2) + 1; int r := n; int i, j; float k;
if l >= 2 then l := l - 1; k := t[l];
else k := t[r]; t[r] := t[1]; r := r - 1;
endif
while r >= 2 do
```

$$r \geq 2, 2l \leq n+1, r+3 \leq n, 2l+2r+1 \leq 3n, l \geq 1, r \leq n$$

```
  i := l; j := 2 * i;
  while j <= r do
```

$$r \geq 2, 2l \leq n+1, r+3 \leq 2n, l \geq 1, r \leq n, 2i = j, l \leq i,$$
$$2i+6l+r+18 \leq 12n, j \leq r, 2l+2r+1 \leq 3n, 4i+2l+1 \leq 2r+3n$$

```
    if j <= r - 1
```

$$r \geq 2, 2l \leq n+1, r+3 \leq 2n, l \geq 1, r \leq n, 2i = j, l \leq i,$$
$$2i+6l+r+18 \leq 12n, j \leq r-1, 2l+2r+1 \leq 3n, 4i+2l+1 \leq 2r+3n$$

```
       and t[j] < t[j+1] then j := j+1; endif
    if k >= t[j] then break; endif
```

$$r+3 \leq 2n, l \geq 1, r \leq n, j \leq 2i+1, 2i \leq j, l \leq i, j \leq r, 2l+2r+1 \leq 3n$$

```
    t[i] := t[j]; i := j; j := 2 * j;
  endwhile
```

$$j+2 \leq 2i+r, 2j+2l \leq 4i+n+1, r+3 \leq 2n, l \geq 1, r \leq n,$$
$$7j+6l+r+18 \leq 12i+12n, 2i \leq j \leq 2i+1, l \leq i, 2l+2r+1 \leq 3n,$$
$$8j+2l+1 \leq 12i+2r+3n$$

```
  t[i] := k;
  if l >= 2 then l := l - 1; k := t[l];
  else k := t[r]; t[r] := t[1]; r := r - 1;
  endif
  t[1] := k;
endwhile
```

THE DOUBLE DESCRIPTION METHOD BY MOTZKIN ET AL.

Constraint Representation

- If $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{a} \neq \mathbf{0}$, and $b \in \mathbb{R}$, the linear inequality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ defines a closed affine half-space.
- All *closed polyhedra* can be expressed as the conjunction of a finite number of such constraints.

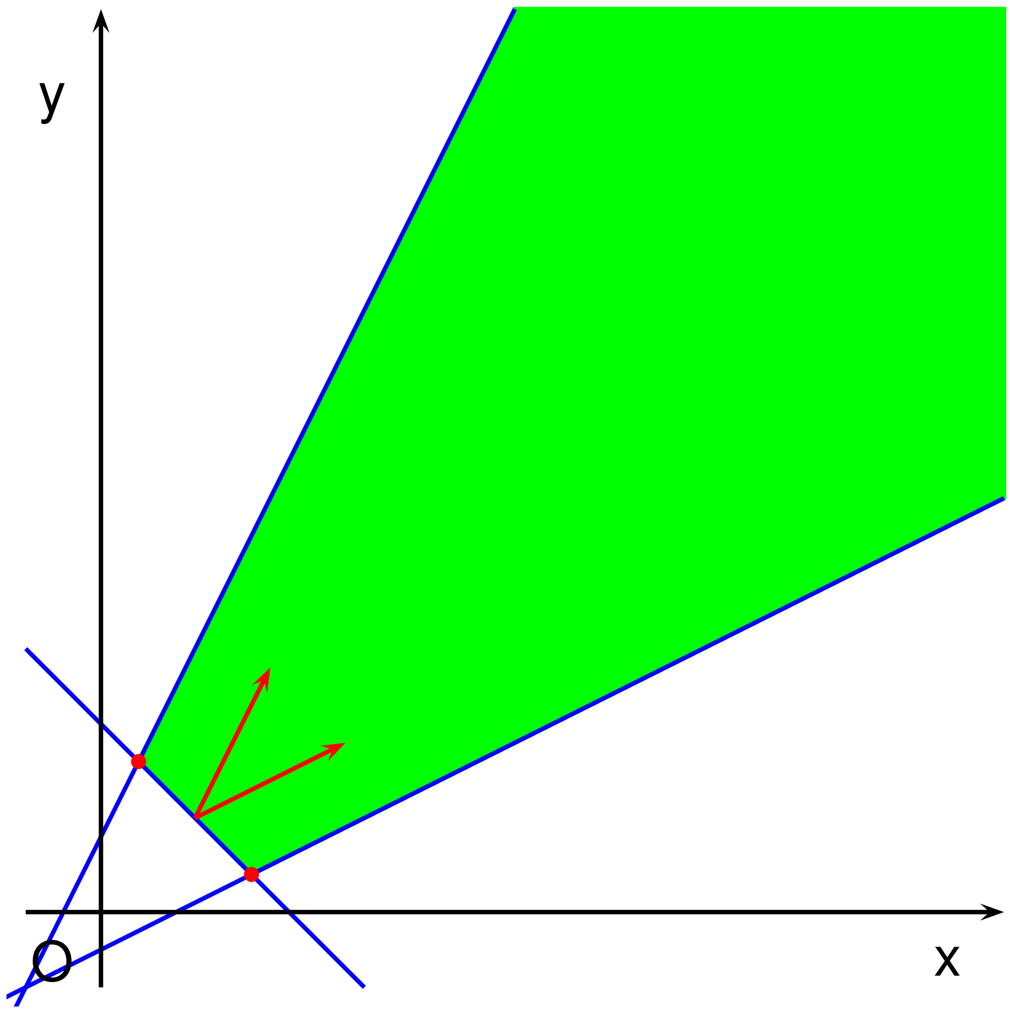
Generator representation

- If $\mathcal{P} \subseteq \mathbb{R}^n$, a *point of* $\mathcal{P}$ is any $\boldsymbol{p} \in \mathcal{P}$.
- If $\mathcal{P} \subseteq \mathbb{R}^n$ and $\mathcal{P} \neq \varnothing$, a vector $\boldsymbol{r} \in \mathbb{R}^n$ such that $\boldsymbol{r} \neq \mathbf{0}$ is a *ray of* $\mathcal{P}$ iff for each point $\boldsymbol{p} \in \mathcal{P}$ and each $\lambda \in \mathbb{R}_+$, we have $\boldsymbol{p} + \lambda \boldsymbol{r} \in \mathcal{P}$.
- All *closed polyhedra* can be expressed as

$$\left\{ R\boldsymbol{\rho} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\middle|\; \boldsymbol{\rho} \in \mathbb{R}^r_+, \boldsymbol{\pi} \in \mathbb{R}^p_+, \textstyle\sum_{i=1}^p \pi_i = 1 \right\}$$

  where $R \in \mathbb{R}^{n \times r}$ is a matrix having rays of the polyhedron as columns and $P \in \mathbb{R}^{n \times p}$ has points of the polyhedron for its columns.

Example: Double Description



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

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

The Double Description Method (cont'd)

Constraint representation

- Special case: $n = 0$ and $\mathcal{P} = \varnothing$.
- The equality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ defines an affine hyperplane...
  - ...that is equivalent to the pair $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ and $\langle -\boldsymbol{a}, \boldsymbol{x} \rangle \geq -b$.
- If $\mathcal{C}$ is a finite set of constraints we call it *a system of constraints* and write $\operatorname{con}(\mathcal{C})$ to denote the polyhedron it describes.

Generator representation

- Note: $P = \varnothing$ if and only if $\mathcal{P} = \varnothing$.
- Note: points are not necessarily vertices and rays are not necessarily extreme.
- We call $\mathcal{G} = (R, P)$ *a system of generators* and write $\operatorname{gen}(\mathcal{G})$ to denote the polyhedron it describes.

DD Pairs and Minimality

Representing a polyhedron both ways

- Let $\mathcal{P} \subseteq \mathbb{R}^n$. If $\mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G}) = \mathcal{P}$, then $(\mathcal{C}, \mathcal{G})$ is said to be a DD pair for $\mathcal{P}$.

Minimality of the representations

- $\mathcal{C}$ is in minimal form if there does not exist $\mathcal{C}' \subset \mathcal{C}$ such that $\mathrm{con}(\mathcal{C}') = \mathcal{P}$;
- $\mathcal{G} = (R, P)$ is in minimal form if there does not exist $\mathcal{G}' = (R', P') \neq \mathcal{G}$ such that $R' \subseteq R$, $P' \subseteq P$ and $\mathrm{gen}(\mathcal{G}') = \mathcal{P}$;
- the DD pair $(\mathcal{C}, \mathcal{G})$ is in minimal form if $\mathcal{C}$ and $\mathcal{G}$ are both in minimal form.

But, wait a minute. . .

. . . why keeping two representations for the same object?

ADVANTAGES OF THE DUAL DESCRIPTION METHOD

Some operations are more efficiently performed on constraints

- ➜ Intersection is implemented as the union of constraint systems.
- ➜ Adding constraints (of course).
- ➜ Relation polyhedron-generator (subsumes or not).

Some operations are more efficiently performed on generators

- ➜ Convex polyhedral hull (poly-hull): union of generator systems.
- ➜ Adding generators (of course).
- ➜ Projection (i.e., removing dimensions).
- ➜ Relation polyhedron-constraint (disjoint, intersects, includes ...).
- ➜ Finiteness (boundedness) check.
- ➜ Time-elapse.

Some operations are more efficiently performed with both

- ➜ Inclusion and equality tests.
- ➜ Widening.

Further Advantages of the Dual Description Method

The principle of duality

- ➜ Systems of constraints and generators enjoy a quite strong and useful duality property.
- ➜ Very roughly speaking:
  - ➜ the constraints of a polyhedron are (almost) the generators of the *polar* of the polyhedron;
  - ➜ the generators of a polyhedron are (almost) the constraints of the polar of the polyhedron;
  - ➜ the polar of the polar of a polyhedron is the polyhedron itself.

$\Longrightarrow$ Computing constraints from generators is the same problem as computing generators from constraints.

The algorithm of Motzkin-Chernikova-Le Verge

- ➜ Solves both problems yielding a minimized system…
- ➜ …and can be implemented so that the source system is also minimized in the process.

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.

PPL Features

Portability across different computing platforms

- written in standard C++;
- but the the client application needs not be written in C++.

Absence of arbitrary limits

- arbitrary precision integer arithmetic for coefficients and coordinates;
- all data structures can expand automatically (in amortized constant time) to any dimension allowed by the available virtual memory.

Complete information hiding

- the internal representation of constraints, generators and systems thereof need not concern the client application;
- implementation devices such as the *positivity constraint* are invisible from outside;
- all the matters regarding the ϵ-representation encoding of NNC polyhedra are also invisible from outside.

PPL FEATURES: HIDING PAYS

Expressivity

- '`X + 2*Y + 5 >= 7*Z`' and '`ray(3*X + Y)`' is valid syntax both for the C++ and the Prolog interfaces;
- we expect the planned Objective Caml, Java and Mercury interfaces to be as friendly as these;
- even the C interface refers to concepts like linear expression, constraint and constraint system
  - (not to their possible implementations such as vectors and matrices).

Failure avoidance and detection

- illegal objects cannot be created easily;
- the interface invariants are systematically checked.

Efficiency

- can systematically apply incremental and lazy computation techniques.

PPL Features: Laziness and Incrementality

Dual description

- ➜ we may have a constraint system, a generator system, or both;
- ➜ in case only one is available, the other is recomputed only when it is convenient to do so.

Minimization

- ➜ the constraint (generator) system may or may not be minimized;
- ➜ it is minimized only when convenient.

Saturation matrices

- ➜ when both constraints and generators are available, some computations record here the relation between them for future use.

Sorting matrices

- ➜ for certain operations, it is advantageous to sort (lazily and incrementally) the matrices representing constraints and generators.

PPL Features: Support for Robustness

```
void complex_function(PH& ph1, const PH& ph2 ...) {
  try {
    start_timer(max_time_for_complex_function);
    complex_function_on_polyhedra(ph1, ph2 ...);
    stop_timer();
  }
  catch (Exception& e) { // Out of memory or timeout...
    BoundingBox bb1, bb2;
    ph1.shrink_bounding_box(bb1);
    ph2.shrink_bounding_box(bb2);
    complex_function_on_bounding_boxes(bb1, bb2 ...);
    ph1 = Polyhedron(bb1);
  }
}
```

- If $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{a} \neq \mathbf{0}$, and $b \in \mathbb{R}$, the linear strict inequality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle > b$ defines an open affine half-space;
- when strict inequalities are allowed in the system of constraints we have polyhedra that are not necessarily closed: NNC polyhedra.
- A fundamental feature of the DD method: the ability to represent polyhedra both by constraints and generators.
- But what are the generators for NNC polyhedra?
- Previous works/implementations did not offer a satisfactory answer.
- By decoupling the user interface from the details of the particular implementation, it is possible to provide an intuitive generalization of the concept of generator system.
- The key step is the introduction of a new kind of generators: closure points:
  - a vector $\boldsymbol{c} \in \mathbb{R}^n$ is a *closure point* of $S \subseteq \mathbb{R}^n$ if and only if $\boldsymbol{c} \in \mathbb{C}(S)$.
- Only the PPL provides, today, this level of support for NNC polyhedra.

MORE PPL FEATURES

Sophisticated widening techniques

- ➜ The first widening operator on convex polyhedra beating the standard one (after 25 years of its introduction).
- ➜ Widening with tokens: an improvement over the delayed widening technique.

The finite powerset construction

- ➜ A generic construction that upgrades an abstract domain by allowing for the exact representation of finite disjunctions of its elements.
- ➜ The PPL offers a generic implementation that can be applied to polyhedra, bounding boxes, octagons, grids, . . .
- ➜ Moreover, this comes with generic widening techniques (implementation in progress, paper to appear at VMCAI'04);
  - ➜ when instantiated on finite powersets of polyhedra, these provide the first widening operators on that domain!

PPL Coming Features

Support for special classes of polyhedra

- A first implementation of bounded differences and octagons is ready;
- a second, more refined implementation will be ready by Q1 2004.
- Partial implementations of intervals and bounding boxes exist: they are waiting for someone to finish them.
- Distinctive features are (beyond the ones already mentioned for the entire library) the tight and smooth integration of all the polyhedra classes and refined widening operators.

Grids and $\mathbb{Z}$-Polyhedra

- A new domain of grids is under development; including support for
  - rational as well as integer values,
  - directions where values will be unrestrained.
- A $\mathbb{Z}$-Polyhedron, which is the intersection of a polyhedron and a grid, will be added once we have the grid domain in the PPL.

NOT THE CONCLUSION

- Convex polyhedra are the basis for several abstractions used in static analysis and computer-aided verification of complex and sometimes mission critical systems.
- For that purposes an implementation of convex polyhedra must be firmly based on a clear theoretical framework and written in accordance with sound software engineering principles.
- In this talk we have presented some of the most important ideas that are behind the Parma Polyhedra Library.
- The Parma Polyhedra Library is free software released under the GPL: code and documentation can be downloaded and its development can be followed at `http://www.cs.unipr.it/ppl/`.

Don't Ask What the PPL Can Do for You; Ask What You Can Do for the PPL (I)

Research work to improve the PPL

- ➜ efficient implementation of polyhedra operations;
- ➜ positive polyhedra;
- ➜ normal forms;
- ➜ widening and narrowing . . .

Research work using the PPL

- ➜ absence of buffer overflows for C and C++ programs;
- ➜ analysis of (machine- and hand-generated) assembly programs;
- ➜ argument size relations for functional and logic programs;
- ➜ optimization of array checks in Java programs;
- ➜ verification of communication and synchronization protocols;
- ➜ verification of linear hybrid systems . . .

Don't Ask What the PPL Can Do for You; Ask What You Can Do for the PPL (II)

Small/medium projects

- internationalization of the library using `gettext`;
- better regression testing with `dejagnu`;
- better STL iterators to go through constraint and generator systems;
- more efficient construction of linear expressions, constraints and generators using expression templates;
- efficient serialization of the various numerical abstractions;
- implementation of the extrapolation operators of Henzinger et al.;
- efficient implementation of convexity recognition of the union of polyhedra (algorithms of Bemporad et al.);
- implementation of a "robust polyhedron" class;
- complete the implementation of the watchdog library;
- implementation of cartesian factoring (Halbwachs et al.) ...

Don't Ask What the PPL Can Do for You; Ask What You Can Do for the PPL (III)

Medium/big projects

- ➜ experiment with different implementations of unlimited precision integers (e.g., `purenum`);
- ➜ complete the implementation of the interval library;
- ➜ Java interface;
- ➜ O'Caml interface;
- ➜ Mercury interface;
- ➜ web-based demo (full of bells and whistles);
- ➜ incorporate the library into various analysis tools;
- ➜ implement some variant of the simplex algorithm;
- ➜ implement cutting-plane methods (Gomory, Chvátal, ...);
- ➜ complete the implementation of the Ask-and-Tell construction;
- ➜ ...