Parma Polyhedra Library Developments and Graphite

R. BAGNARA, P. M. HILL, E. ZAFFANELLA

University of Parma, Italy

University of Leeds, United Kingdom

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

Plan of the Talk

1. The Parma Polyhedra Library
2. Implemented Features Possibly Interesting for Graphite
   1. Finite Powerset Domains
   2. Mixed Integer Programming Problems
   3. Grids
   4. Simplification Using Context
   5. Generalized Intervals
3. Unimplemented Features Possibly Interesting for Graphite
   1. Polyhedra Simplifications
   2. Operations to Capture Bounded Arithmetic
   3. Circular Linear Progressions
   4. Operations to Capture Floating Point Computations
   5. Polyhedral Sorting
4. Conclusion

Part I
The Parma Polyhedra Library

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 of a wide range of numerical abstractions targeted at abstract interpretation and computer-aided verification.
  - many kinds of boxes (still experimental);
  - BD and octagonal shapes;
  - (not necessarily closed) convex polyhedra;
  - grids (i.e., linear congruences);
  - finite sets of boxes, BDSs, OSs, (NNC) polyhedra;
  - MIP (Mixed Integer Programming) solver.
- Free software released under the GNU General Public License.

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 (used by default) for coefficients and coordinates;
- ➜ optionally, can use machine integers in a 100% safe way;
- ➜ 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* or ϵ*-polyhedra* are 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;
- work on the Objective Caml and Java interfaces aims at making them 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(C_Polyhedron& ph1,
                      const C_Polyhedron& 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...
    Rational_Box rb1(ph1), rb2(ph2);
    complex_function_on_rational_boxes(rb1, rb2 ...);
    ph1 = C_Polyhedron(rb1);
  }
}
```

Part II
Implemented Features Possibly Interesting for Graphite

Finite Powerset Domains

A finite powerset upgrades any abstract domain into a refined one where finite disjunctions of non redundant elements are precisely representable.

The class template `Pointset_Powerset<PS>` is suitable for the instantiation with the “semantic” numerical domains of the PPL: (C or NNC) polyhedra, bounded difference shapes, octagonal shapes, grids and combinations thereof.

A unique feature of this implementation is the provision of a provably correct certificate-based widening operators that lift the widening operators defined on the underlying domain.

Using the `Pointset_Powerset<C_Polyhedron>` domain would allow to simplify the code of CLooG-PPL considerably

Mixed Integer Programming Problems

PPL 0.10 includes a Mixed Integer (Linear) Programming (MIP) solver based on the simplex algorithm and branch-and-bound, and using exact arithmetic.

The MIP solver interface allows for both satisfiability checks and optimization of linear objective functions.

The solver is incremental, in that it allows for the efficient re-optimization of a MIP problem after modification of the objective function and the addition of new constraints.

GRIDS: INTRODUCTION

Numerical information can be summarized in two basic ways:

GRIDS: INTRODUCTION

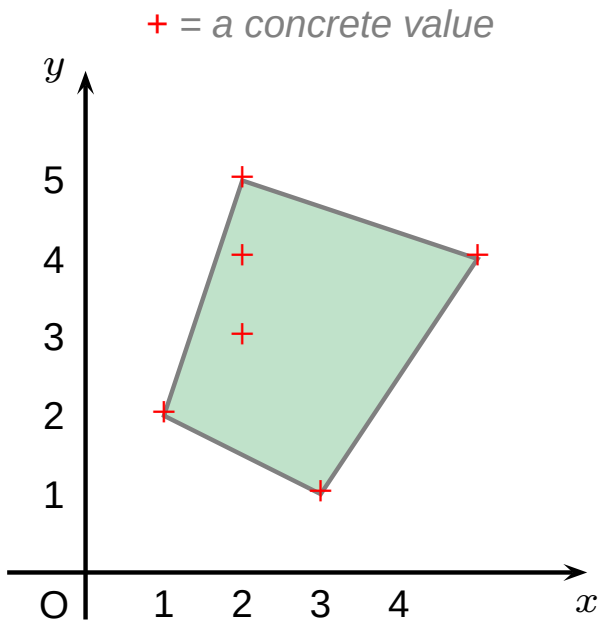
Numerical information can be summarized in two basic ways:

1. by providing outer limits or bounds within which the values lie;

GRIDS: INTRODUCTION

Numerical information can be summarized in two basic ways:

1. by providing outer limits or bounds within which the values lie;

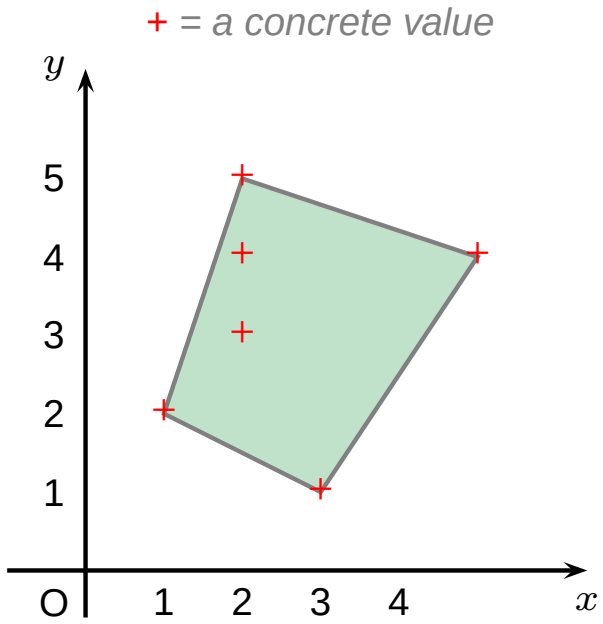


1. Polyhedral domain

GRIDS: INTRODUCTION

Numerical information can be summarized in two basic ways:

1. by providing outer limits or bounds within which the values lie;
2. by providing the pattern of distribution of these values.

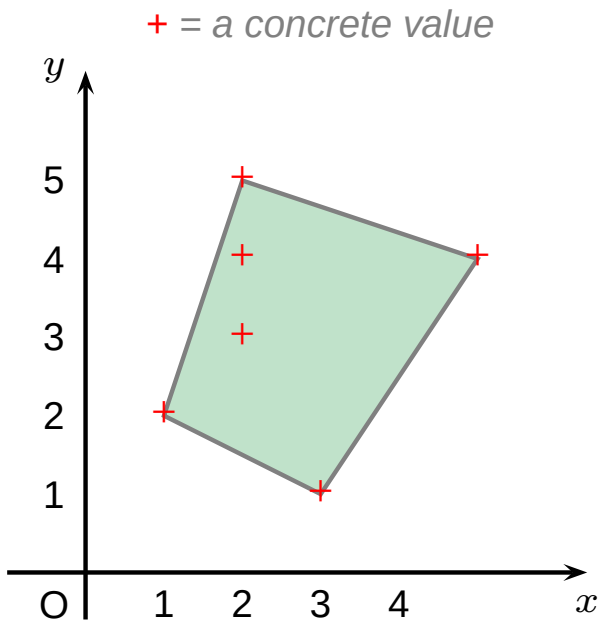


1. Polyhedral domain

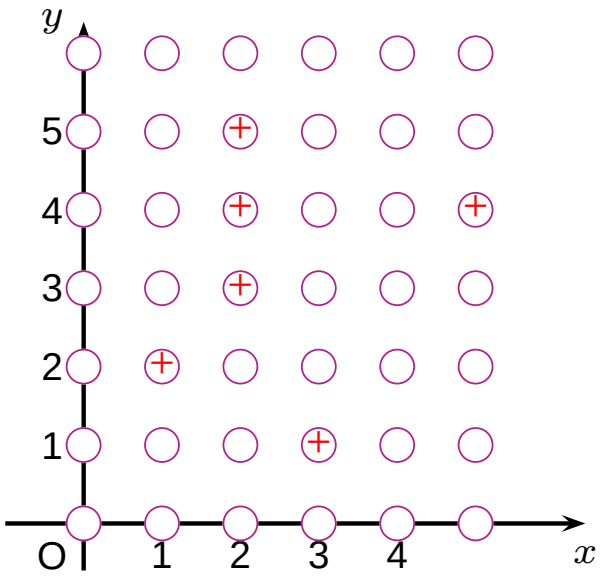
GRIDS: INTRODUCTION

Numerical information can be summarized in two basic ways:

1. by providing outer limits or bounds within which the values lie;
2. by providing the pattern of distribution of these values.



1. Polyhedral domain

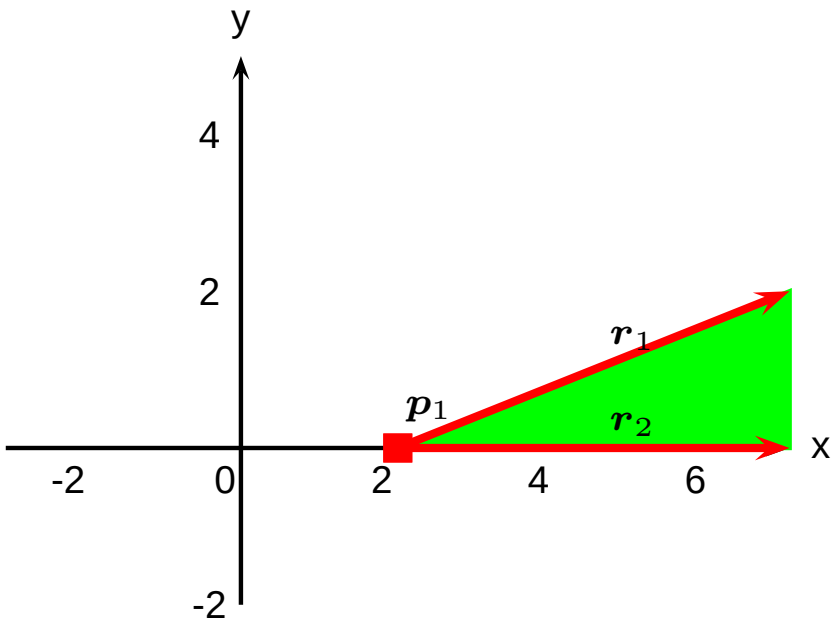


2. Grid domain

Double Representations for Polyhedra and Grids

- A Polyhedron is all the points in a region bounded by hyperplanes;
  the Polyhedron can be represented in 2 ways:
  - a finite set of linear inequalities called constraints
  - a finite set of generators.
- A Grid is a set of equally spaced points;
  the grid can be represented in 2 ways:
  - a finite set of linear congruence relations
  - a finite set of generators.

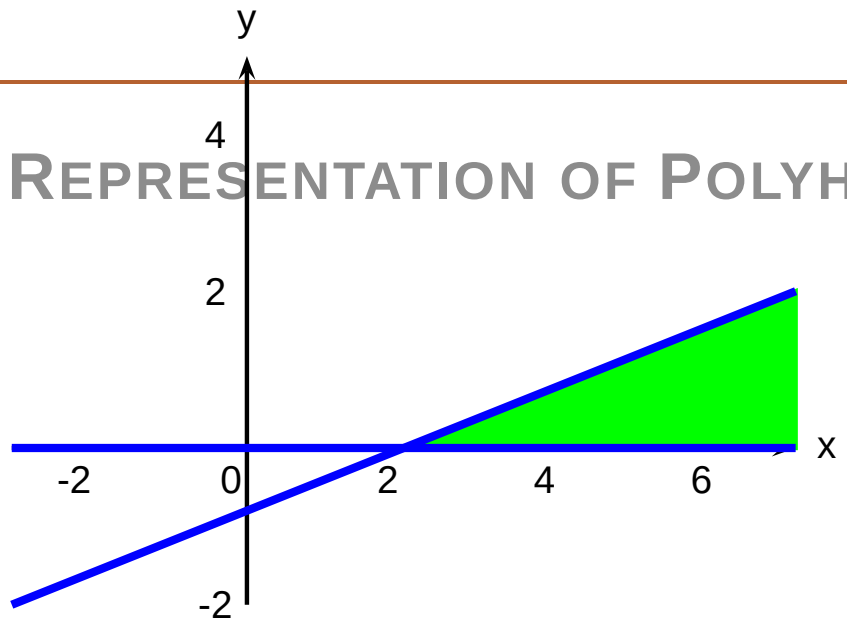
Double Representation of Polyhedra: Generators



The polyhedron can be generated by the point and two rays

$$\boldsymbol{p}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad \boldsymbol{r}_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad \boldsymbol{r}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Double Representation of Polyhedra: Constraints



The polyhedron can also be represented by the constraints

$$y \geq 0, \qquad x - 2y \leq 2.$$

Double Representation of Grids: Congruences (I)

For vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalars $b, f \in \mathbb{R}$, the *congruence relation*

$$\big(\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b \mod f\big)$$

defines the set

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

For example:

- ➜ $(x = 1 \mod 0)$ is the equality $x = 1$;
- ➜ $(x = 1 \mod 2)$ defines the odd integers:
  $\ldots, -1, 1, 3, 5, \ldots$;
- ➜ $(x - y = 0 \mod 2)$ defines the lines:
  $\ldots, x = y - 2, x = y, x = y + 2, x = y + 4, \ldots$.

Double Representation of Grids: Congruences (II)

Consider the following example in two dimensions.

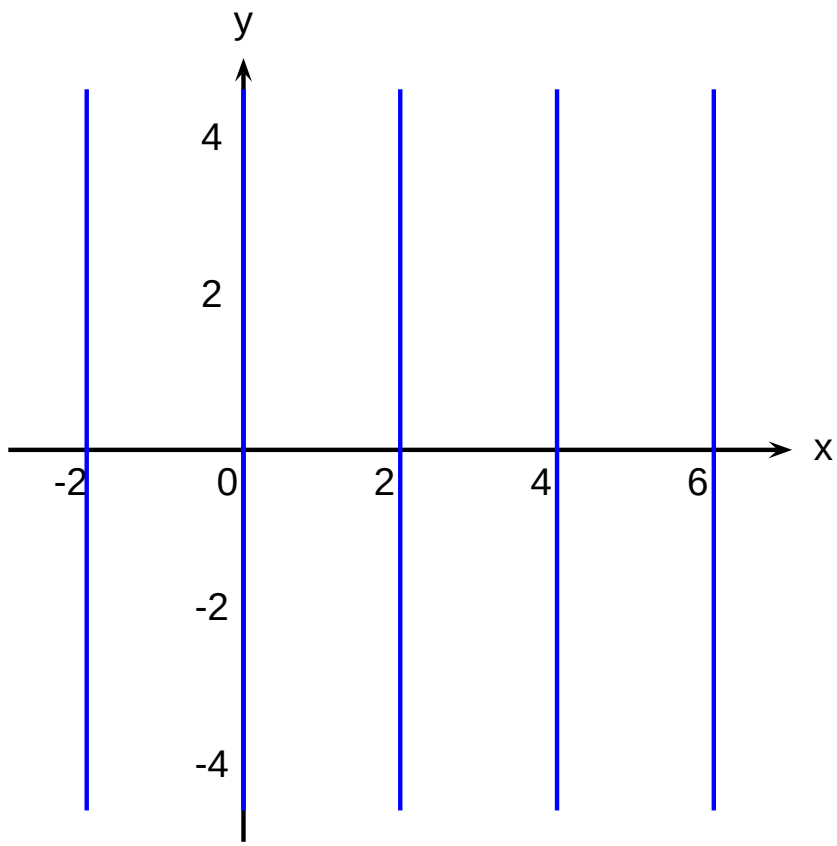
Let $(x = 0 \mod 2)$ be a congruence relation.

Double Representation of Grids: Congruences (II)

Consider the following example in two dimensions.

Let $(x = 0 \mod 2)$ be a congruence relation.

A congruence system



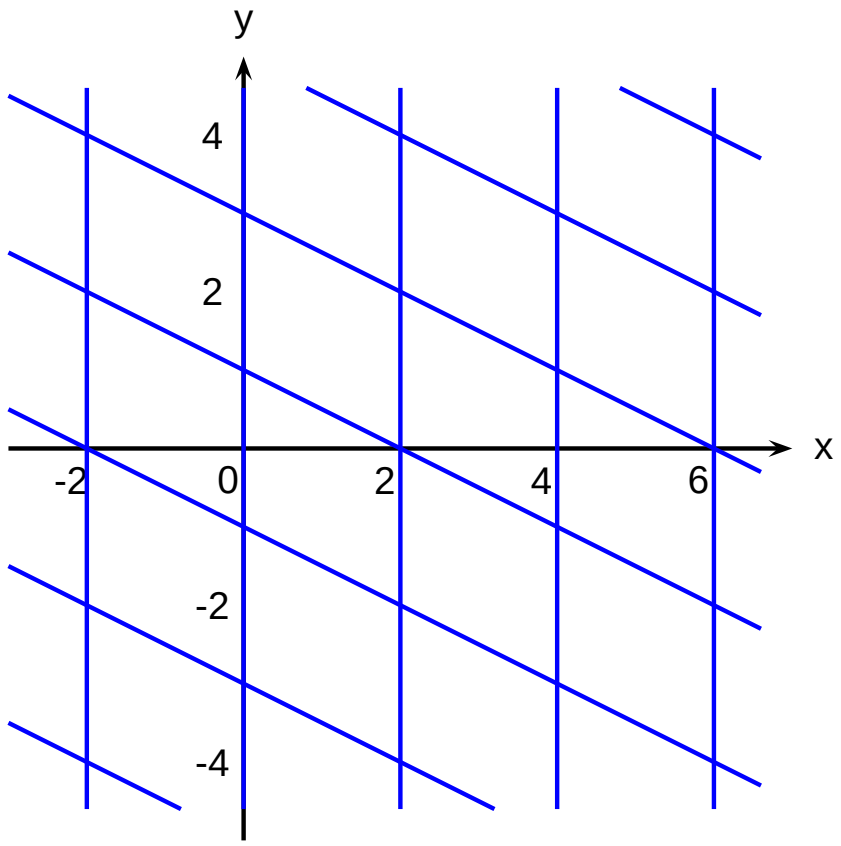
Double Representation of Grids: Congruences (II)

Consider the following example in two dimensions.

Let $(x = 0 \mod 2)$ be a congruence relation.

Now add the congruence relation $(x + 2y = 2 \mod 4)$.

A congruence system



Double Representation of Grids: Congruences (II)

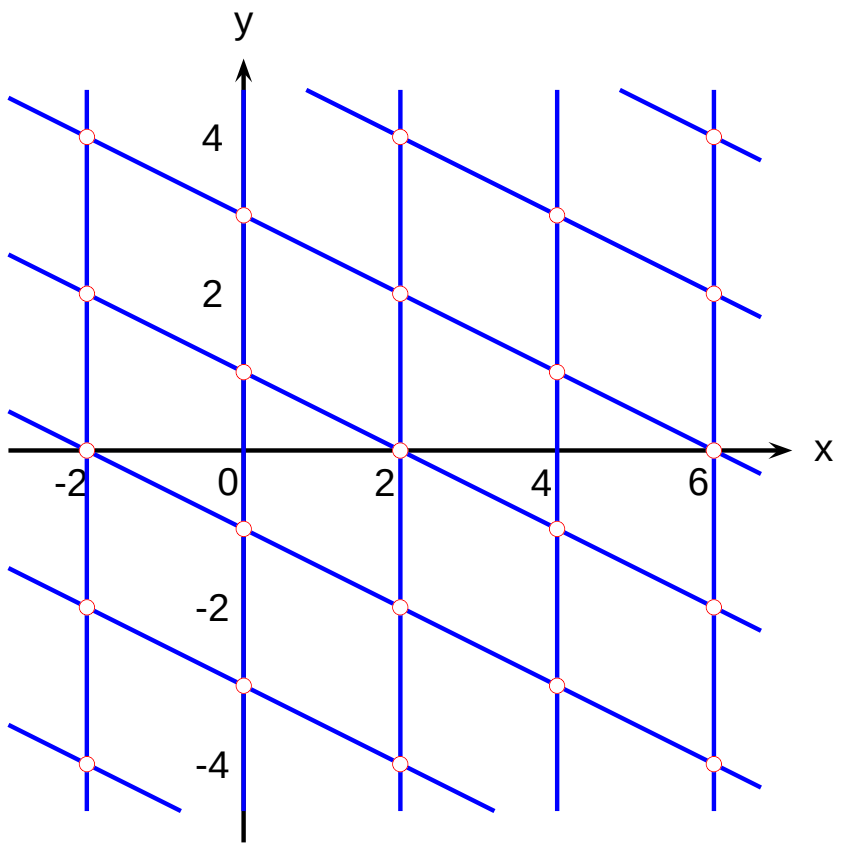
Consider the following example in two dimensions.

Let $(x = 0 \mod 2)$ be a congruence relation.

Now add the congruence relation $(x + 2y = 2 \mod 4)$.

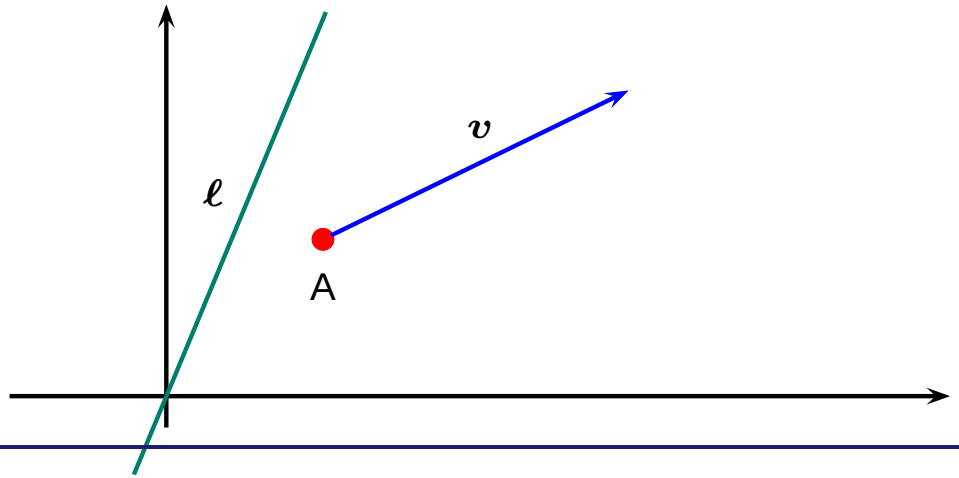
The grid is the set of points that satisfy both congruence relations.

A congruence system



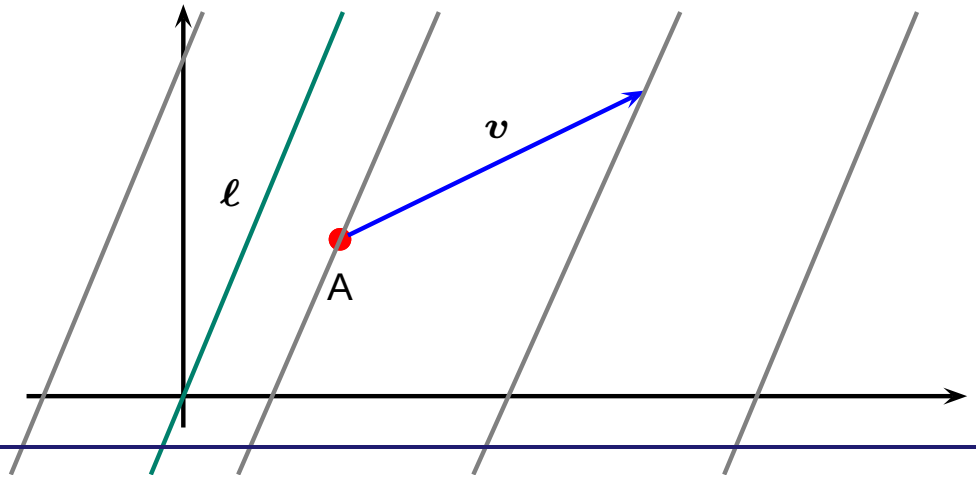
Double Representation of Grids: Generators (I)

- A grid generator is a vector in n-dimensional space.
- It can either be a grid point (A), parameter ($\boldsymbol{v}$) or grid line ($\boldsymbol{\ell}$).
- A grid is the set of points produced by taking
  an integral affine combination of the generating points P
  + an integral combination of the generating parameters Q
  + a linear combination of the generating lines L.



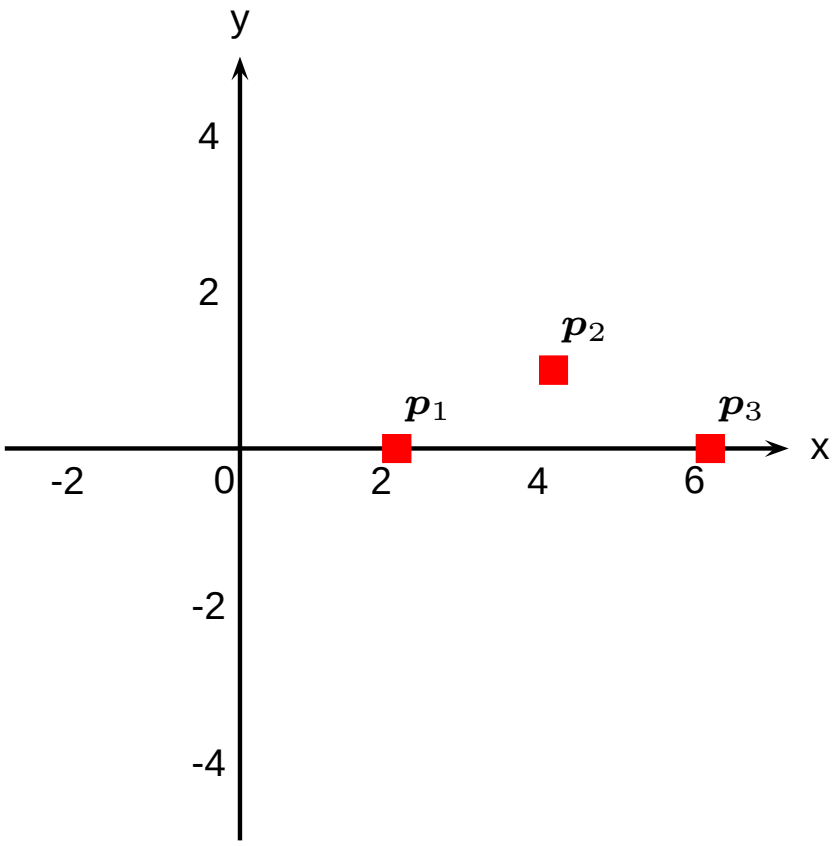
Double Representation of Grids: Generators (I)

- A grid generator is a vector in n-dimensional space.
- It can either be a grid point (A), parameter ($\boldsymbol{v}$) or grid line (ℓ).
- A grid is the set of points produced by taking
  an integral affine combination of the generating points P
  + an integral combination of the generating parameters Q
  + a linear combination of the generating lines L.



Double Representation of Grids: Generators (II)

A Generator System

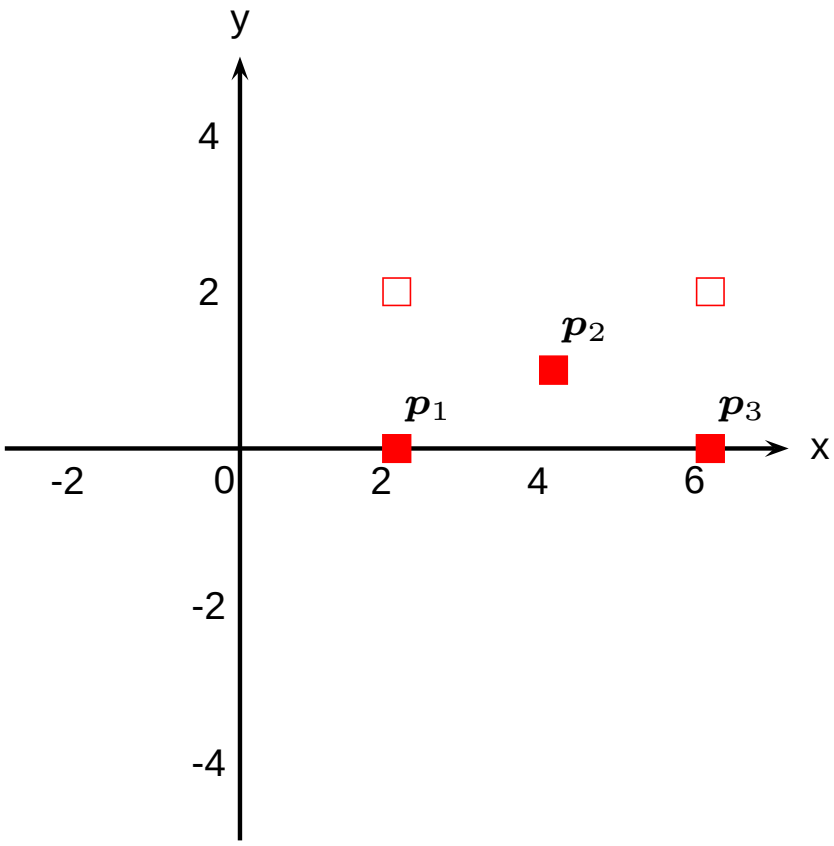


Consider the 3 generating points

$$\boldsymbol{p}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \boldsymbol{p}_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \boldsymbol{p}_3 = \begin{pmatrix} 6 \\ 0 \end{pmatrix}.$$

Double Representation of Grids: Generators (II)

A Generator System



Consider the 3 generating points

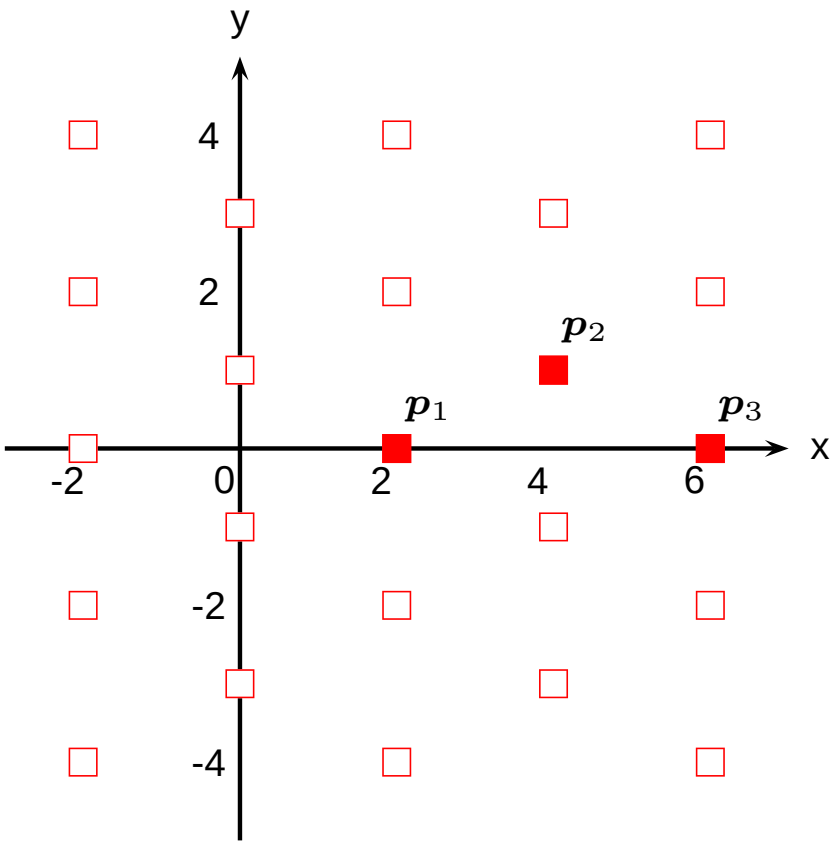
$$p_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, p_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, p_3 = \begin{pmatrix} 6 \\ 0 \end{pmatrix},$$

we can generate points

$$\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 2p_2 - p_3, \begin{pmatrix} 6 \\ 2 \end{pmatrix} = 2p_2 - p_1.$$

Double Representation of Grids: Generators (II)

A Generator System



Consider the 3 generating points

$$\boldsymbol{p}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \boldsymbol{p}_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \boldsymbol{p}_3 = \begin{pmatrix} 6 \\ 0 \end{pmatrix},$$

we can generate points

$$\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 2\boldsymbol{p}_2 - \boldsymbol{p}_3, \begin{pmatrix} 6 \\ 2 \end{pmatrix} = 2\boldsymbol{p}_2 - \boldsymbol{p}_1, \cdots$$

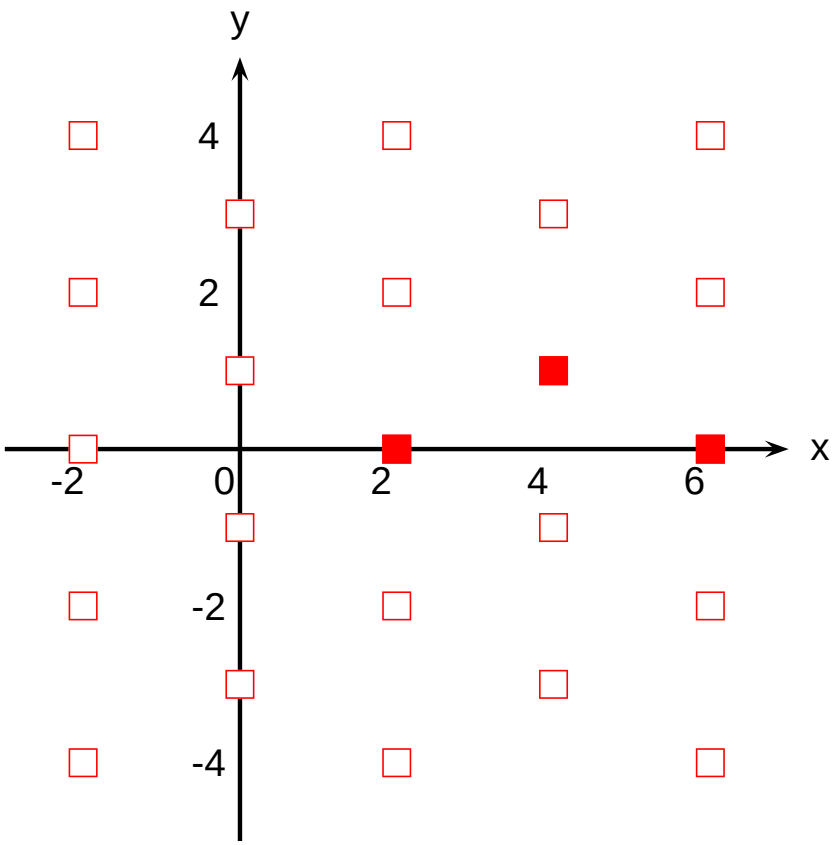
$$\boldsymbol{v} = \pi_1\boldsymbol{p}_1 + \pi_2\boldsymbol{p}_2 + \pi_3\boldsymbol{p}_3,$$

where $\pi_1, \pi_2, \pi_3 \in \mathbb{Z}$

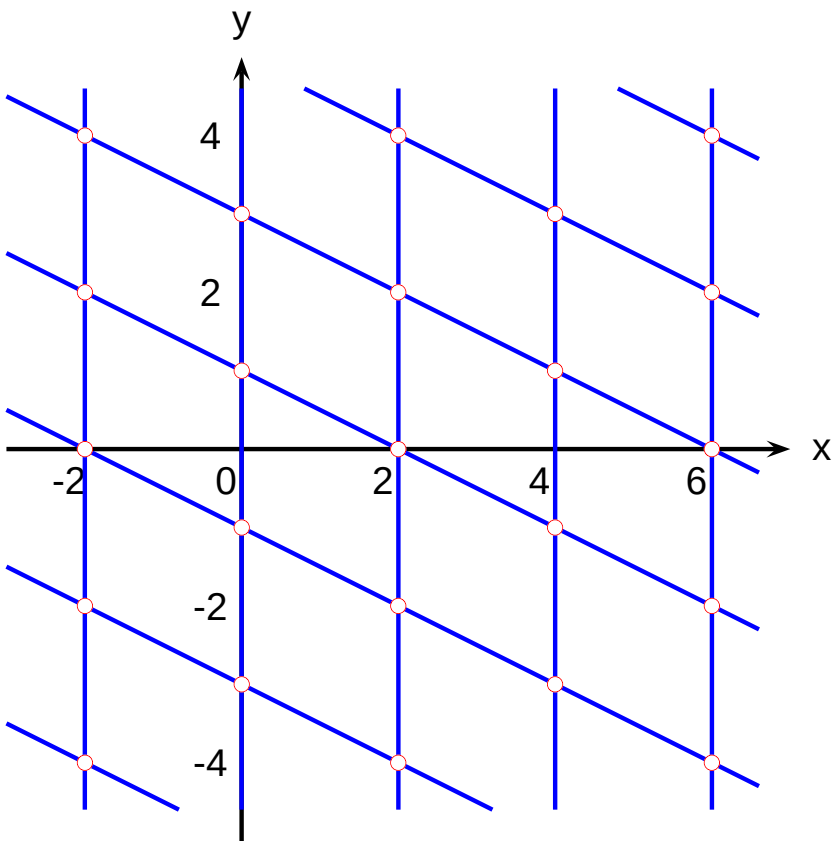
and $\pi_1 + \pi_2 + \pi_3 = 1$.

Double Representation of Grids: Congruences and Generato

A generator system



A congruence system



Simplify Using Context: Specification (I)

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

- ➜ meet-preserving wrt $\mathcal{P}$ using context $\mathcal{Q}$, if $\mathcal{R} \cap \mathcal{Q} = \mathcal{P} \cap \mathcal{Q}$;
- ➜ an enlargement of $\mathcal{P}$, if $\mathcal{R} \supseteq \mathcal{P}$;
- ➜ a simplification wrt $\mathcal{P}$, if $r \leq p$, where r and p are the cardinalities of minimized constraint representations for $\mathcal{R}$ and $\mathcal{P}$.

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q}) \stackrel{\mathrm{def}}{=}$ all meet-preserving enlargement & simplifications of $\mathcal{P}$ using context $\mathcal{Q}$.

Note: $\mathcal{P} \in \mathrm{mpes}(\mathcal{P}, \mathcal{Q})$.

Generalizable to boxes, bounded difference shapes, octagons, etc.

Simplify Using Context: Specification (II)

Let $\mathcal{S}_1 = \{d_1, \ldots, d_m\}$, $\mathcal{S}_2 = \{c_1, \ldots, c_n\}$, $\mathcal{S} = \{s_1, \ldots, s_q\}$ be Omega-reduced powersets of polyhedra. Then $\mathcal{S}$ is said to be:

- ➜ meet-preserving (wrt $\mathcal{S}_1$ using context $\mathcal{S}_2$), if $\mathcal{S} \sqcap \mathcal{S}_2 = \mathcal{S}_1 \sqcap \mathcal{S}_2$;
- ➜ a powerset simplification wrt $\mathcal{S}_1$, if $q \leq m$.
- ➜ a disjunct meet-preserving simplification, if each disjunct in $\mathcal{S}$ has been computed as an mpes of a disjunct of $\mathcal{S}_1$ wrt the context $\mathcal{S}_2$.

$$\forall s_k \in \mathcal{S} : \exists d_i \in \mathcal{S}_1 \,.\, \forall c_j \in \mathcal{S}_2 : s_k \in \mathrm{mpes}(d_i, c_j).$$

Goal: fewer disjuncts, each one as large and simple as possible, while preserving the powerset meet.

Non-goal: merge different, possibly overlapping disjuncts so as to further simplify the result (easily leads to inefficiencies).

Simplify Using Context: Implementation (I)

Base domain level (i.e., not powersets):

- ➜ Filter away special cases: $\mathcal{P}$ or $\mathcal{Q}$ is empty;
- ➜ Compute intersection $\mathcal{Z} = \mathcal{P} \cap \mathcal{Q}$;
- ➜ If $\mathcal{Z} = \varnothing$, find a minimal set of constraints from $\mathcal{P}$ making $\mathcal{Q}$ empty:
  - ➜ heuristics: select $\mathcal{P}$ constraints cutting more of $\mathcal{Q}$ generators;
  - ➜ the emptiness test is based on the incremental MIP solver.
- ➜ If $\mathcal{Z} \neq \varnothing$, drop redundant constraints of $\mathcal{P}$:
  - ➜ the redundancy test is based on saturation with respect to the generators of $\mathcal{Q}$.

Simplification Using Context: Implementation (II)

Key step for the powerset level.

➜ Simplify single d with respect to the powerset context $\mathcal{S}_2 = \{c_1, \ldots, c_n\}$:

```
s := universe polyhedron;
for c in S_2 do
  tighter_c := c intersection s;
  simpl_d_c := simplify d using tighter_c;
  s := s intersection simpl_d_c;
end for;
return s;
```

➜ Context tightening remembers the constraints that have already been marked as non-redundant in a previous iteration.

Simplification Using Context: Implementation (III)

- ➜ Filter away special cases:
  1. the powerset $\mathcal{S}_1$ to be simplified is empty;
  2. the powerset context $\mathcal{S}_2$ is empty or a singleton.
- ➜ Handle general case:

```
S := emptyset;
for d in S_1 do
   if d intersects any c in S_2 then
      s := simplify d using S_2;
      insert s into S;
   endif;
return S;
```

- ➜ Note: intersection test is actually performed during simplification of d.

Generalized Intervals

As of version 0.10, the PPL provides a very general flexible and unfinished implementation of intervals:

- ➜ closed and (optionally) open boundaries;
- ➜ several possible choices for the boundaries:
  - any bounded precision native integer type;
  - any bounded precision floating point type;
  - unbounded integer or rational types, as provided by GMP;
- ➜ (completely orthogonal) support for generic restrictions: this allows to capture, e.g.,
  - plain intervals of real numbers,
  - intervals of integer numbers and
  - modulo intervals (proposed by Nakanishi, Joe, Polychronopoulos and Fukuda at PACT ’99).

Part III
Unimplemented Features Possibly Interesting for Graphit

Polyhedra Simplifications

- In paper

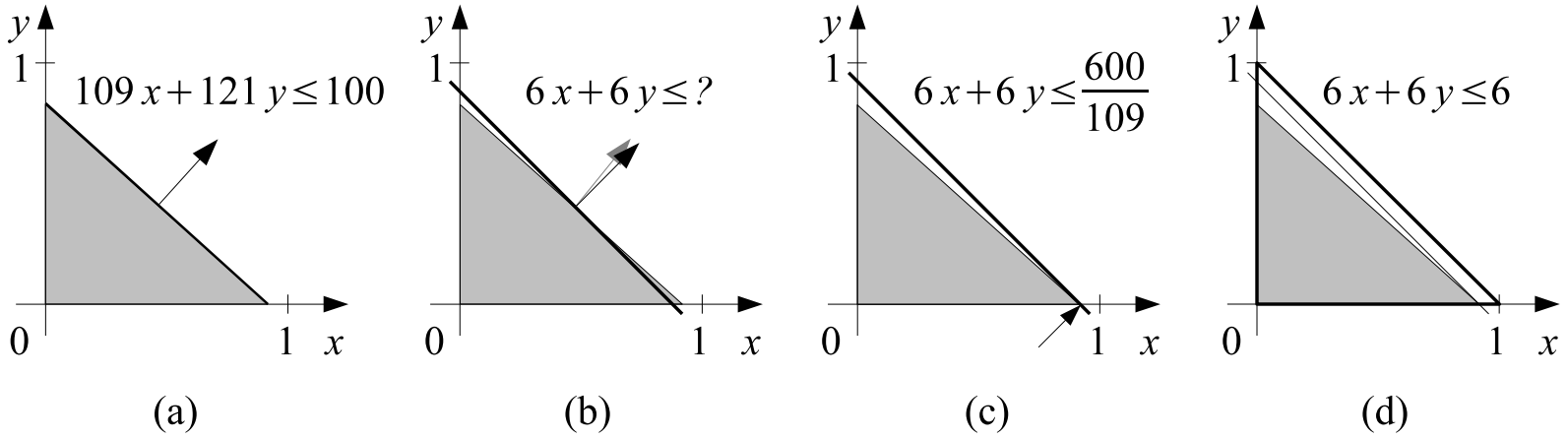
  G. Frehse.
  PHAVer: Algorithmic verification of hybrid systems past HyTech.
  In *Proceedings of HSCC 2005*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273, Zürich, Switzerland, 2005.

  two safe “simplifications” of polyhedra are presented:

  1. Decreasing the size of the (integer and unbounded) coefficients in the constraint representation.
  2. Decreasing the number of constraints.

- Can we do something similar on the generator representation (when the constraint representation is not available)?
- Can we simplify both representations (when they are both available)?

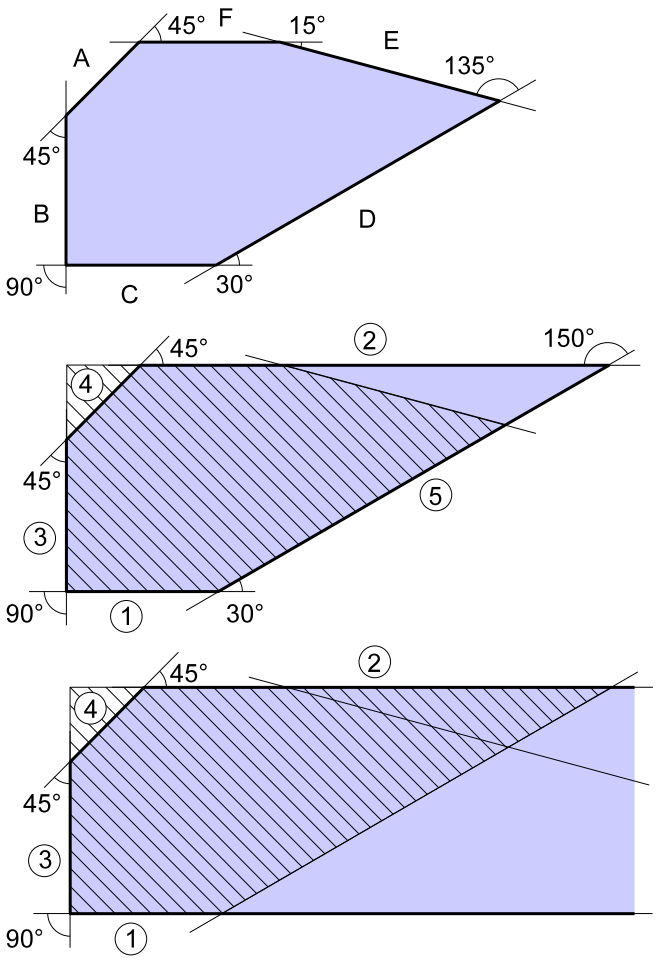
Limiting the Number of Bits in Coefficients



1. Truncate bits of homogeneous coefficients.
2. Solve an LP to push out the plane.
3. Snap to next integer.

(Pictures courtesy of Goran Frehse.)

Limiting the Number of Constraints



Example of 6-constraints polyhedron approximated with 5 and 4 constraints, respectively.

Hashed approximations obtained by volumetric deconstruction.

Shaded approximations obtained by angle-based reconstruction.

(Pictures courtesy of Goran Frehse.)

“Decremental” Double Description

- Double Description algorithms *à la* Chernikova support incremental modifications of the representations:
  - After the addition of a new constraint (resp., generator), we can efficiently recompute the two minimized descriptions.
- What about decremental modifications?
  - What is the best algorithm allowing for the efficient removal of a constraint or a generator?
  - Trivial observation: the removal of a constraint (resp., generator) maintains the minimality of the constraint (resp., generator) description.
- The removal of some of its constraints is the simplest way of approximating from above a polyhedron: it is important that such a “simplification” does not incur an efficiency penalty.

Bounded Arithmetic and Nonlinear Expressions

At present the PPL does not offer direct support for bounded arithmetic, for the approximation of bitwise or shift operations, for the linearization of nonlinear expressions.

We have some (Prolog!) prototypical code that:

- ➜ implements precise interval approximations of these operations;
- ➜ linearizes (with a high degree of precision) nonlinear expressions.

Part or all of this code could eventually be incorporated into the PPL, but this requires a significant amount of work.

Circular Linear Progressions (I)

This is a numerical abstraction proposed by Sen and Srikant at MEMOCODE '07 (which is strictly more precise than the strided intervals proposed by Reps, Balakrishnan and Lim at PEPM '06).

A CLP is a triple $C = (l, u, \delta)$ where:

- $l \in \mathbb{Z}(n)$ is the starting point;
- $u \in \mathbb{Z}(n)$ is the ending point;
- $\delta \in \mathbb{N}(n)$ is the increment.

The semantics of C is given by

$$\gamma(C) \stackrel{\text{def}}{=} \left\{ a_i \;\middle|\; a_i = l +_n i\delta, i \in \mathbb{N}, i \leq s, s = \min\{ s \in \mathbb{N} \mid a_s = u \} \right\}.$$

Circular Linear Progressions (II)

Operators are defined to correctly capture all arithmetic and bitwise operations, as well as logical and arithmetic shifts

We are studying a variation of the original proposal that is suitable for tracking unsigned variables.

Of course, the resulting abstract domain is non-relational. The original proposal includes a combination with Karr's domain for tracking linear equalities.

Operations to Capture Floating Point Computations

Currently, the PPL does not provide direct support for the approximation of floating point computations.

We are studying two proposals:

- ➜ the one by Miné, presented at ESOP ’04;
- ➜ the one by Simon, presented at NSAD ’05.

It is not yet clear which is best.

Both seem to suffer from coefficient explosion: we conjecture that coupling them with the polyhedral simplifications mentioned above could provide a satisfactory solution.

Polyhedral Sorting

- A polyhedra comparison routine used in CLooG:

  `int compare(p, q, level, params, dims)`

- Ad-hoc interpretation of space dimensions:
  - first $dims - params$ are iteration domains (loop indices);
  - last $params$ are parameters (invariant loop bounds);
  - $level$ is a fixed iteration domain.
- (Sort of) Lexicographic comparison wrt constraints on $level$.
- No need to code this routine inside the PPL: current PPL public interface allows PPL users (e.g., CLooG developers) to implement the routine without incurring significant overhead.

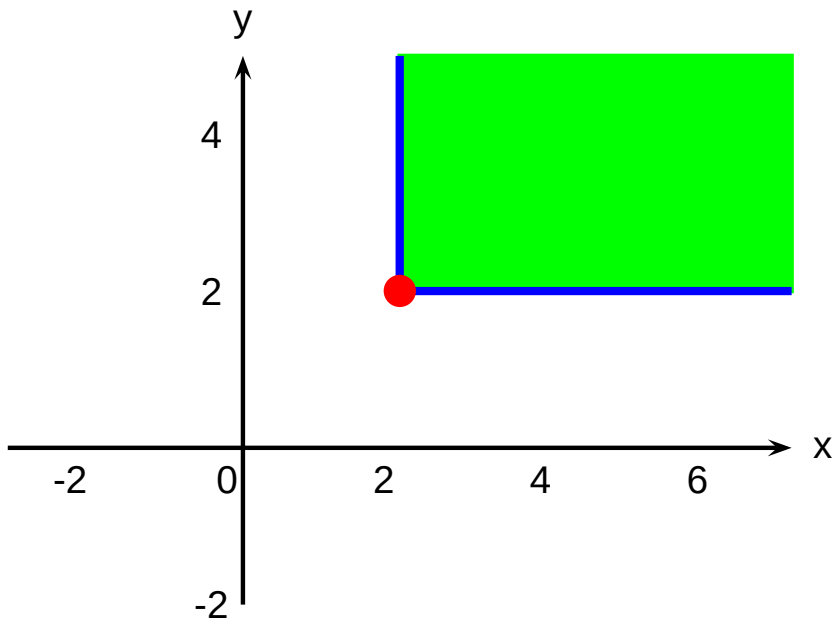
Conclusion

The PPL team, within the limits imposed by the extreme scarcity of resources, will do its best to support Graphite.

Your feedback is very valuable: please feel free to write to `ppl-devel@cs.unipr.it` to report bugs or simple imperfections, to suggest new features or to criticize the existing ones.

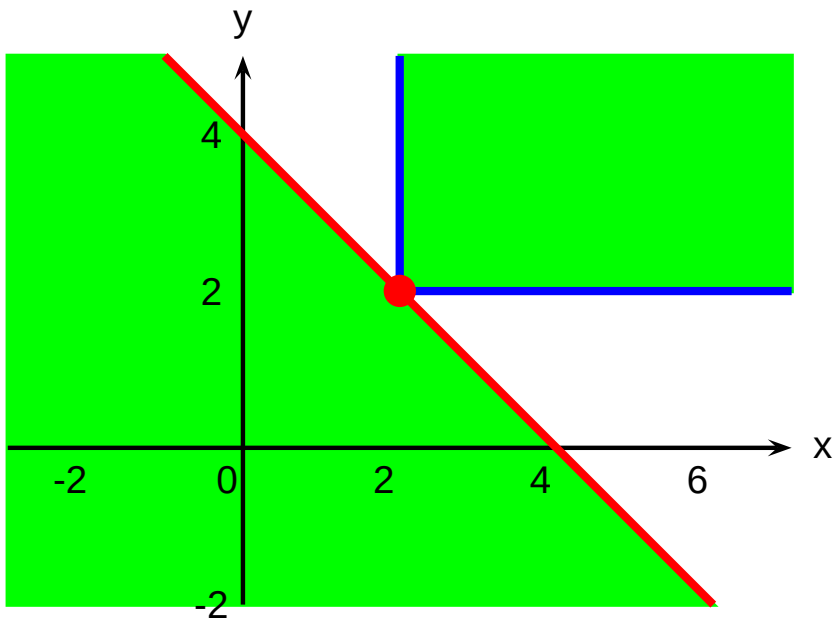
We would like to anticipate the possible scalability problems and to study them in depth: can you provide us with tough testcases?

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q})$ With Infinitely Many Maximal Elements (I)



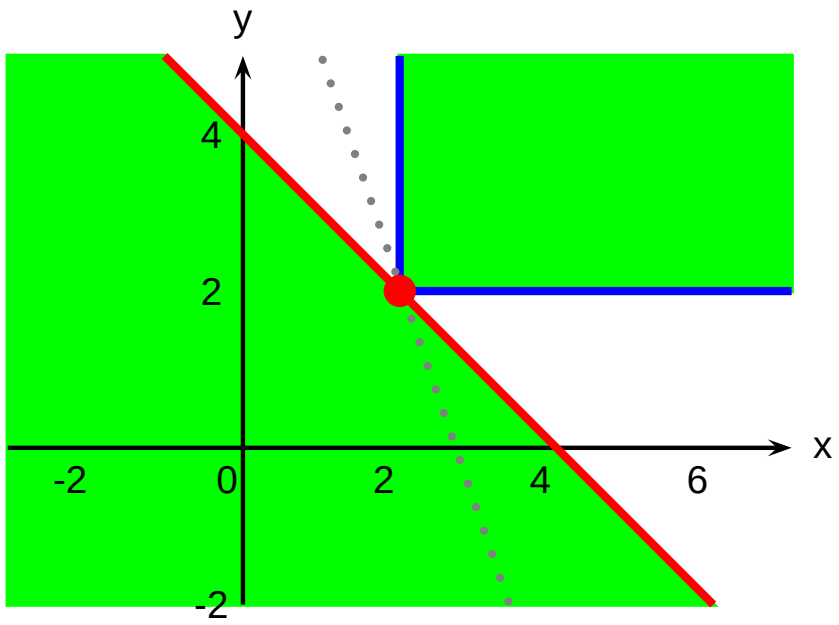
Simplify $\mathcal{P} = \{x = 2, y = 2\}$ using context $\mathcal{Q} = \{x \geq 2, y \geq 2\}$?

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q})$ With Infinitely Many Maximal Elements (II)



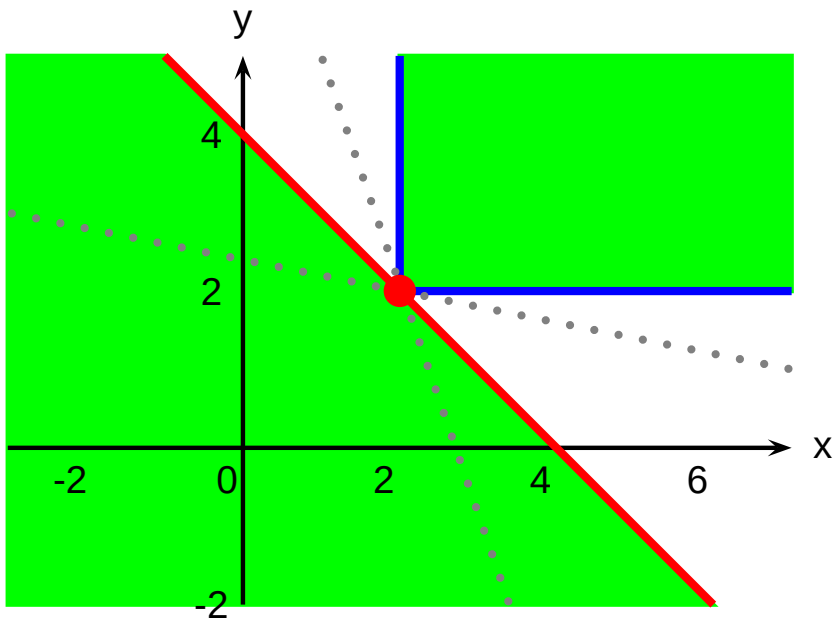
Simplify $\mathcal{P} = \{x = 2, y = 2\}$ using context $\mathcal{Q} = \{x \geq 2, y \geq 2\}$?

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q})$ With Infinitely Many Maximal Elements (III)



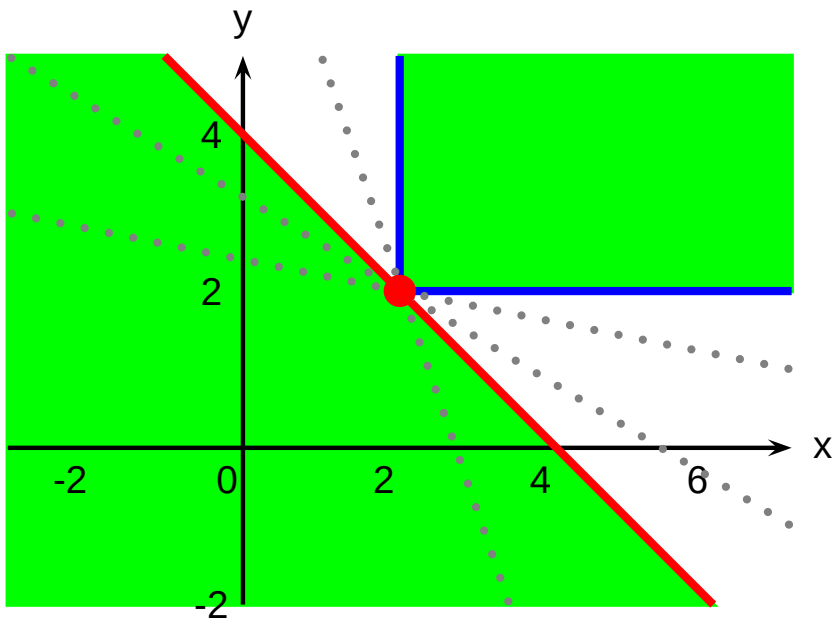
Simplify $\mathcal{P} = \{x = 2, y = 2\}$ using context $\mathcal{Q} = \{x \geq 2, y \geq 2\}$?

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q})$ With Infinitely Many Maximal Elements (iv)



Simplify $\mathcal{P} = \{x = 2, y = 2\}$ using context $\mathcal{Q} = \{x \geq 2, y \geq 2\}$?

$\mathrm{mpes}(\mathcal{P}, \mathcal{Q})$ With Infinitely Many Maximal Elements (v)



Simplify $\mathcal{P} = \{x = 2, y = 2\}$ using context $\mathcal{Q} = \{x \geq 2, y \geq 2\}$?

Example Where PPL Improves wrt PolyLib on Simplificatio

Example is `polylib-5.22.3/Test/general/simpl6.in`:

- Simplify

$$\mathcal{P} = \{x = 1, y + 1 = 0, z = 3\}$$

using context

$$\mathcal{Q} = \{x = 1, y + z = 2, 0 \leq z \leq 3\} :$$

- PolyLib expected result:

$$\mathcal{R} = \{y + 1 = 0, z = 3\};$$

- PPL computed result:

$$\mathcal{R}' = \{z = 3\}.$$