Abstract Interpretation and the Parma Polyhedra Library: from Theory to Practice and Vice Versa

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 Problem
2. Formal Program Verification Methods
3. An Example
4. Topologically Closed Convex Polyhedra
5. The Double Description Method by Motzkin et al.
6. The Parma Polyhedra Library: Theory Meets Practice
7. The Return Trip: from Practice Back to Theory
   1. Widening Operators
   2. Finite Powerset Domains
   3. NNC Polyhedra
8. Current and Future Development Plans

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 underflow): 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.

Example: The Concrete Semantics

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

while x <= 100 do
```

$(x, y) \in S \in \wp(\mathbb{R}^2)$

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

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

  endif

endwhile
```

Concrete domain:

$$\langle \wp(\mathbb{R}^2), \subseteq, \varnothing, \mathbb{R}^2, \cup, \cap \rangle.$$

Concrete Semantics:

$$S \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F} = \mathcal{F}^{\omega}(\varnothing).$$

Example: The Concrete Semantics

```
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: The Concrete Semantics

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

$\{(0,0)\}$

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

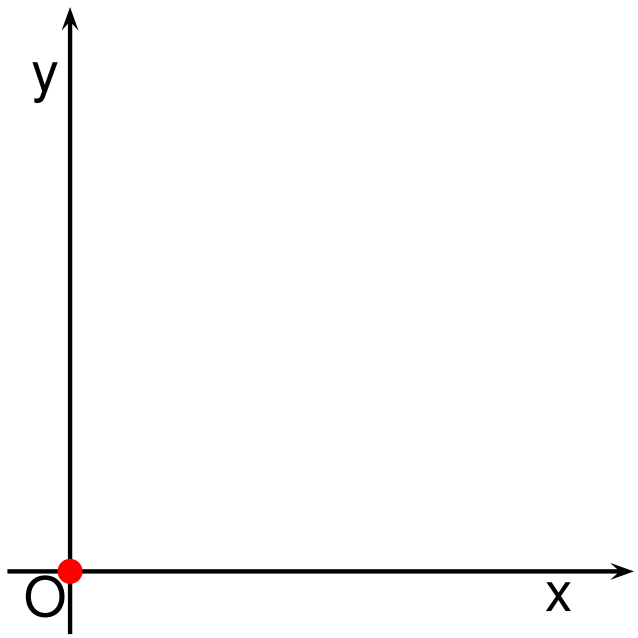
$\varnothing$

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

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

  endif

endwhile
```



Example: The Concrete Semantics

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

$\{(0,0)\}$

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

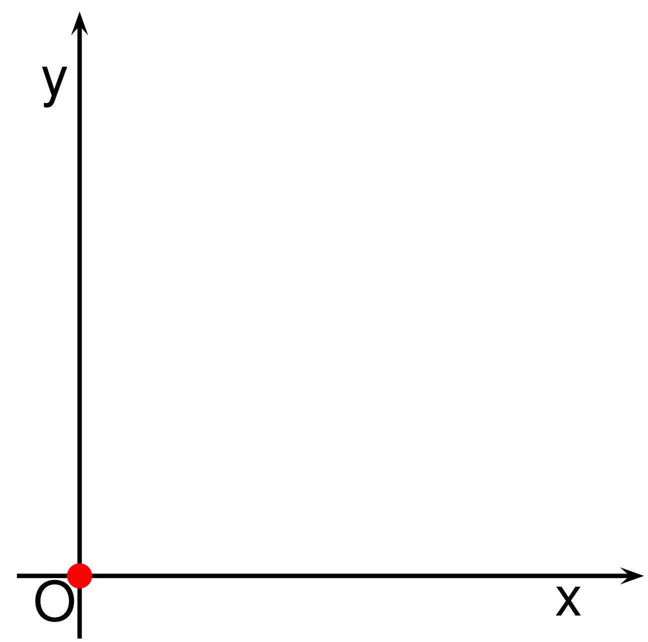
$\{(0,0)\}$

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

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

  endif

endwhile
```

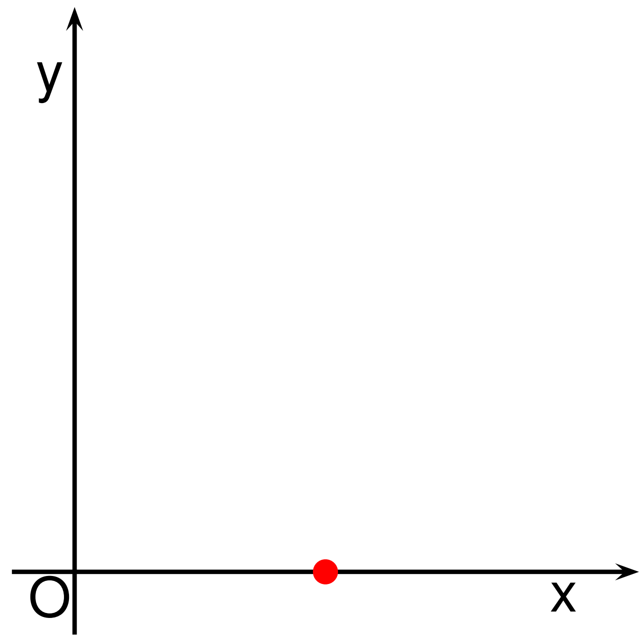


Example: The Concrete Semantics

```
x := 0; y := 0;
```
$\{(0,0)\}$
```
while x <= 100 do
```
$\{(0,0)\}$
```
  read(b);
  if b then x := x+2
```
$\{(2,0)\}$
```
  else x := x+1; y := y+1;

  endif

endwhile
```



Example: The Concrete Semantics

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

$\{(0, 0)\}$

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

$\{(0, 0)\}$

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

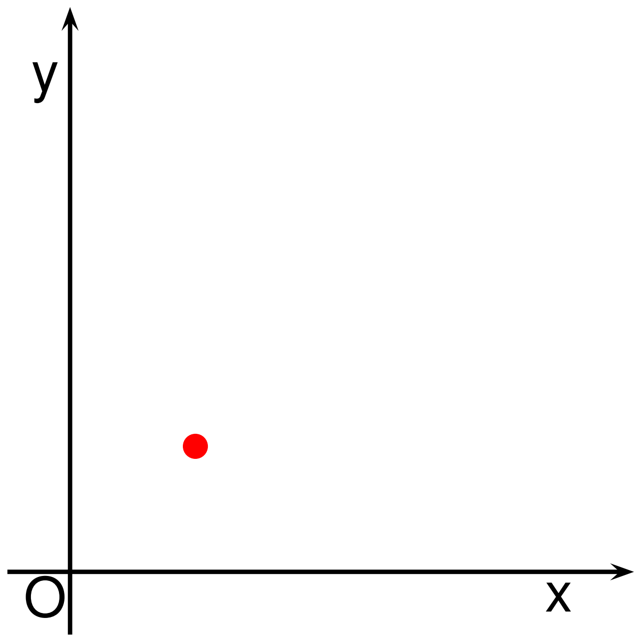
$\{(2, 0)\}$

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

$\{(1, 1)\}$

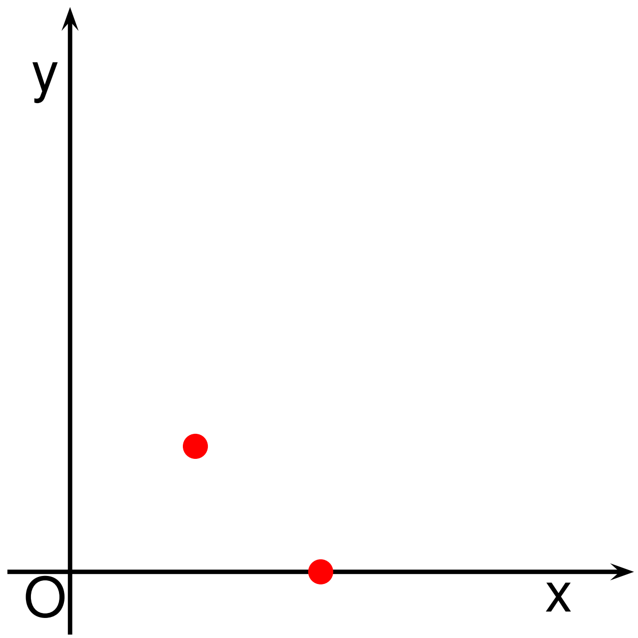
```
  endif

endwhile
```



Example: The Concrete Semantics

```
x := 0; y := 0;
```
$\{(0,0)\}$
```
while x <= 100 do
```
$\{(0,0)\}$
```
  read(b);
  if b then x := x+2
```
$\{(2,0)\}$
```
  else x := x+1; y := y+1;
```
$\{(1,1)\}$
```
  endif
```
$\{(1,1),(2,0)\}$
```
endwhile
```



Example: The Concrete Semantics

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

$\{(0,0)\}$

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

$\{(0,0),(1,1),(2,0)\}$

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

$\{(2,0)\}$

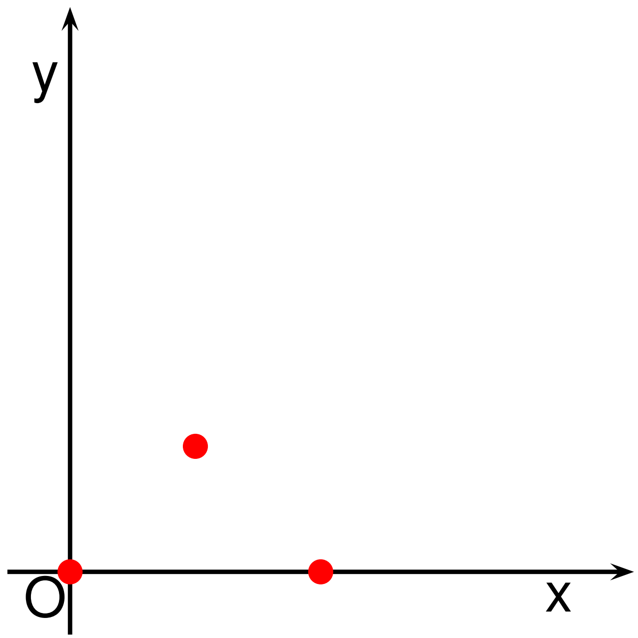
```
  else x := x+1; y := y+1;
```

$\{(1,1)\}$

```
  endif
```

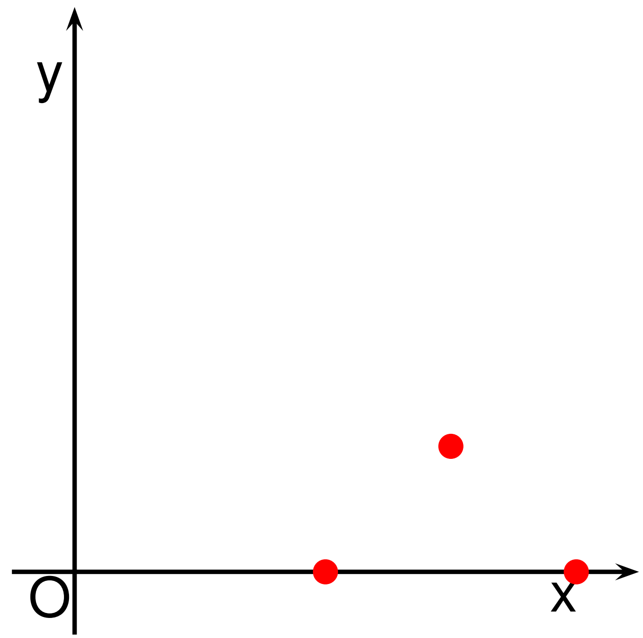
$\{(1,1),(2,0)\}$

```
endwhile
```



Example: The Concrete Semantics

```
x := 0; y := 0;
```
$\{(0,0)\}$
```
while x <= 100 do
```
$\{(0,0),(1,1),(2,0)\}$
```
  read(b);
  if b then x := x+2
```
$\{(2,0),(3,1),(4,0)\}$
```
  else x := x+1; y := y+1;
```
$\{(1,1)\}$
```
  endif
```
$\{(1,1),(2,0)\}$
```
endwhile
```



Example: The Concrete Semantics

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

$\{(0,0)\}$

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

$\{(0,0),(1,1),(2,0)\}$

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

$\{(2,0),(3,1),(4,0)\}$

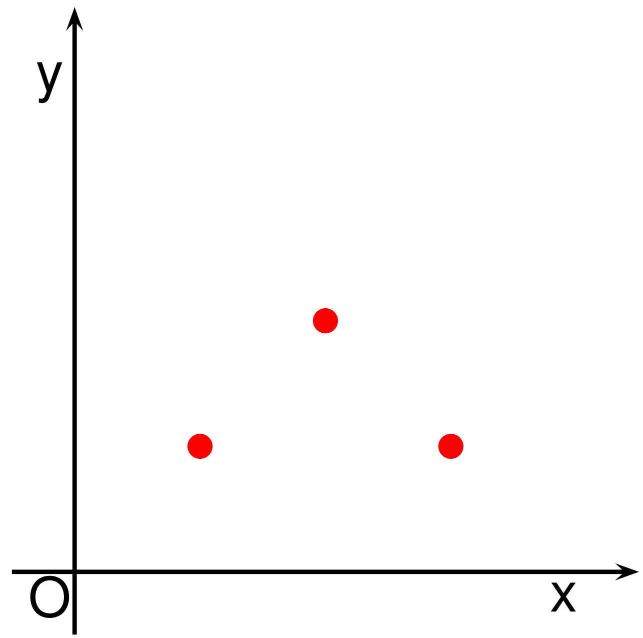
```
  else x := x+1; y := y+1;
```

$\{(1,1),(2,2),(3,1)\}$

```
  endif
```

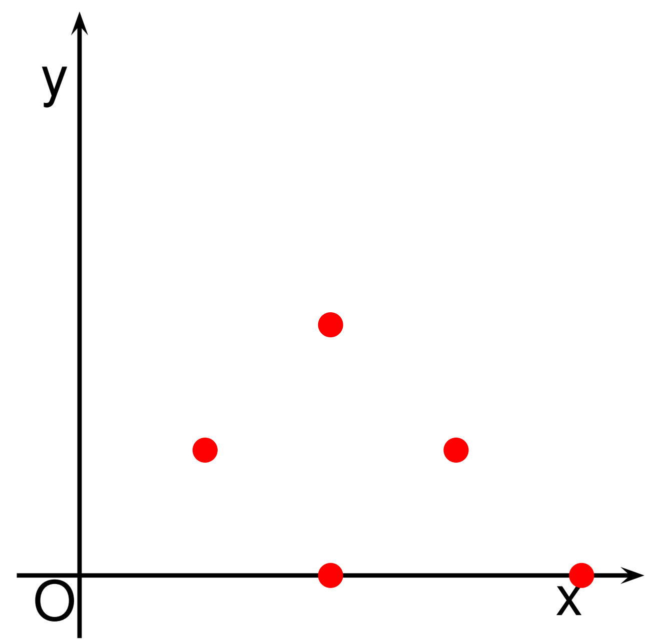
$\{(1,1),(2,0)\}$

```
endwhile
```



Example: The Concrete Semantics

```
x := 0; y := 0;
```
$\{(0,0)\}$
```
while x <= 100 do
```
$\{(0,0),(1,1),(2,0)\}$
```
  read(b);
  if b then x := x+2
```
$\{(2,0),(3,1),(4,0)\}$
```
  else x := x+1; y := y+1;
```
$\{(1,1),(2,2),(3,1)\}$
```
  endif
```
$\{(1,1),(2,0),(2,2),(3,1),(4,0)\}$
```
endwhile
```



Example: ... and so on ...

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

$\{(0,0)\}$

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

$\{(0,0),(1,1),(2,0),(2,2),(3,1),(4,0)\}$

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

$\{(2,0),(3,1),(4,0)\}$

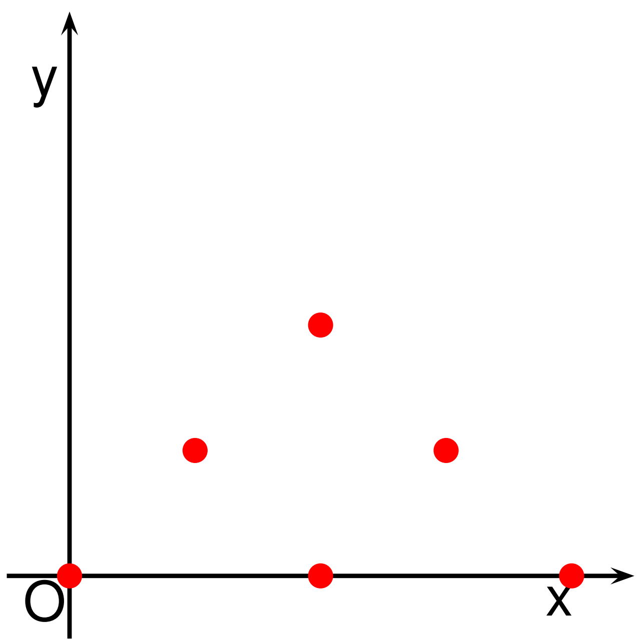
```
  else x := x+1; y := y+1;
```

$\{(1,1),(2,2),(3,1)\}$

```
  endif
```

$\{(1,1),(2,0),(2,2),(3,1),(4,0)\}$

```
endwhile
```



Example: The Abstract Semantics

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

while x <= 100 do
```

$(x, y) \in \mathcal{Q} \in \mathbb{CP}_2$

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

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

  endif

endwhile
```

Abstract domain:

$$\langle \mathbb{CP}_2, \subseteq, \varnothing, \mathbb{R}^2, \uplus, \cap \rangle.$$

Correctness:

$$X \subseteq \mathcal{P} \implies \mathcal{F}(X) \subseteq \mathcal{F}^\sharp(\mathcal{P}).$$

Abstract Semantics:

$$\mathcal{Q} \in \mathrm{postfp}(\mathcal{F}^\sharp).$$

Example: The Abstract Semantics

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

while x <= 100 do
```
$\{1 = 0\}$
```
  read(b);
  if b then x := x+2

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

  endif

endwhile
```

Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

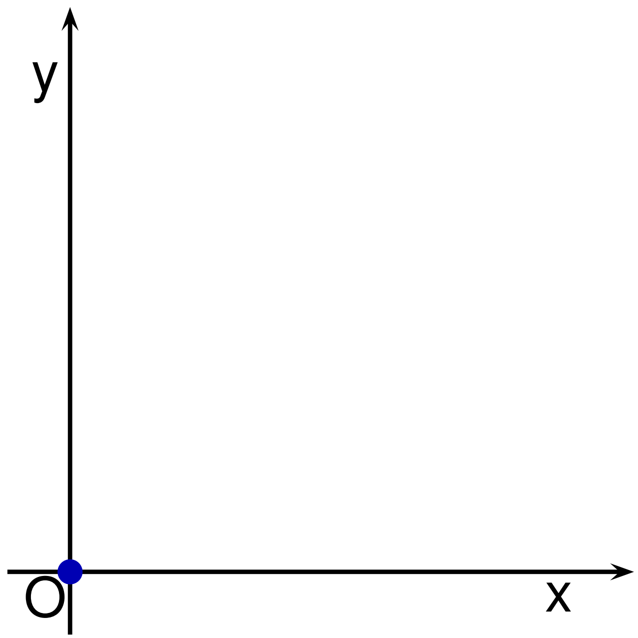
$\{1 = 0\}$

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

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

  endif

endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

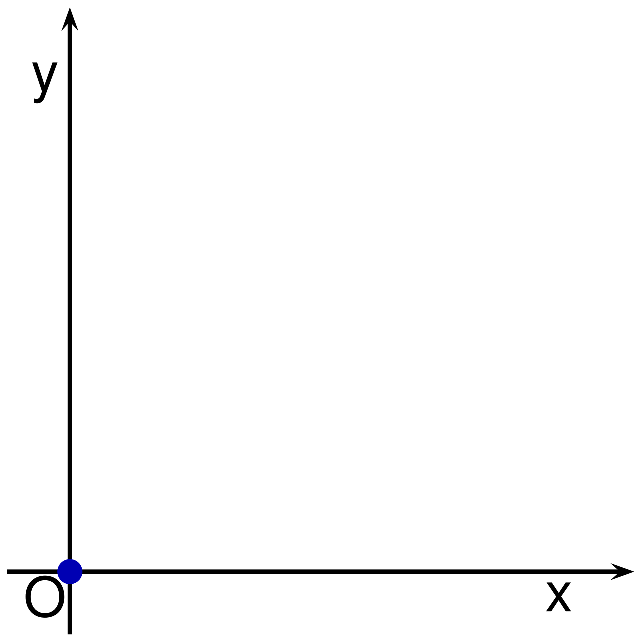
$\{x = 0, y = 0\}$

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

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

  endif

endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{x = 0, y = 0\}$

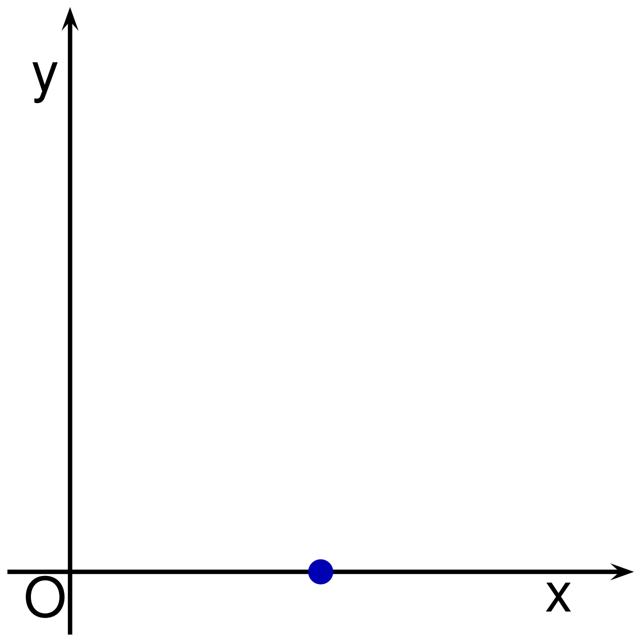
```
  read(b);
  if b then x := x+2
```

$\{x = 2, y = 0\}$

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

  endif

endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{x = 0, y = 0\}$

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

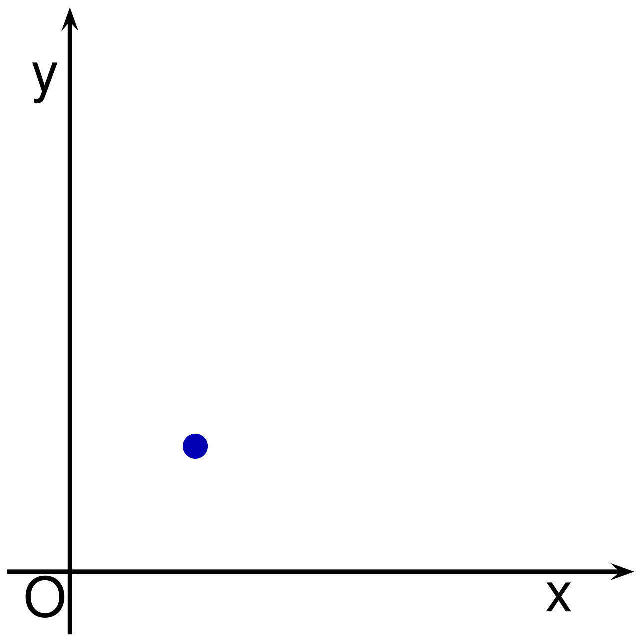
$\{x = 2, y = 0\}$

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

$\{x = 1, y = 1\}$

```
  endif

endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{x = 0, y = 0\}$

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

$\{x = 2, y = 0\}$

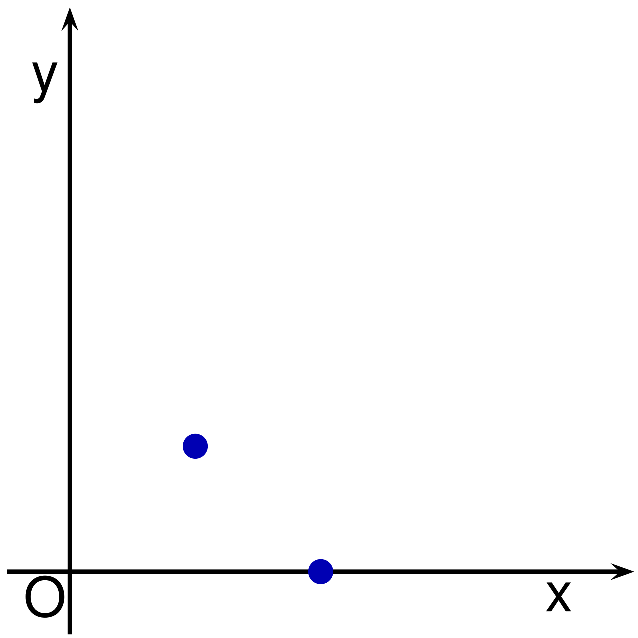
```
  else x := x+1; y := y+1;
```

$\{x = 1, y = 1\}$

```
  endif
```

$\{x = 2, y = 0\} \uplus \{x = 1, y = 1\}$

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{x = 0, y = 0\}$

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

$\{x = 2, y = 0\}$

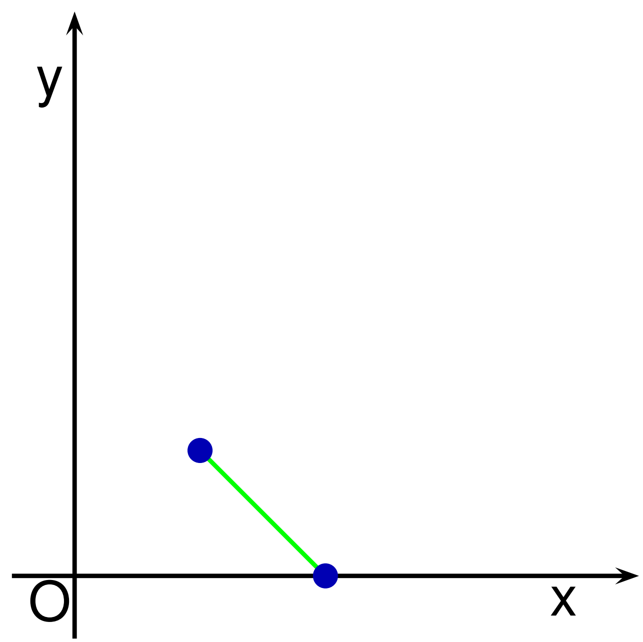
```
  else x := x+1; y := y+1;
```

$\{x = 1, y = 1\}$

```
  endif
```

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

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{x = 0, y = 0\}$

$\uplus \{1 \leq x \leq 2, x + y = 2\}$

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

$\{x = 2, y = 0\}$

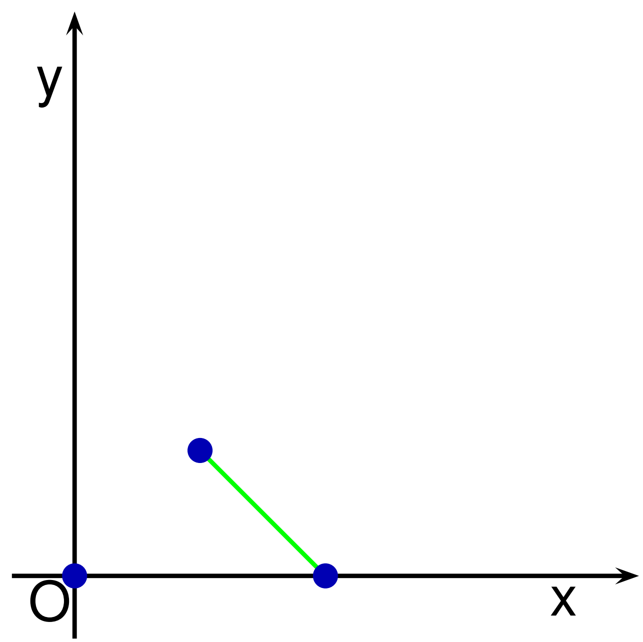
```
  else x := x+1; y := y+1;
```

$\{x = 1, y = 1\}$

```
  endif
```

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

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

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

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

$\{x = 2, y = 0\}$

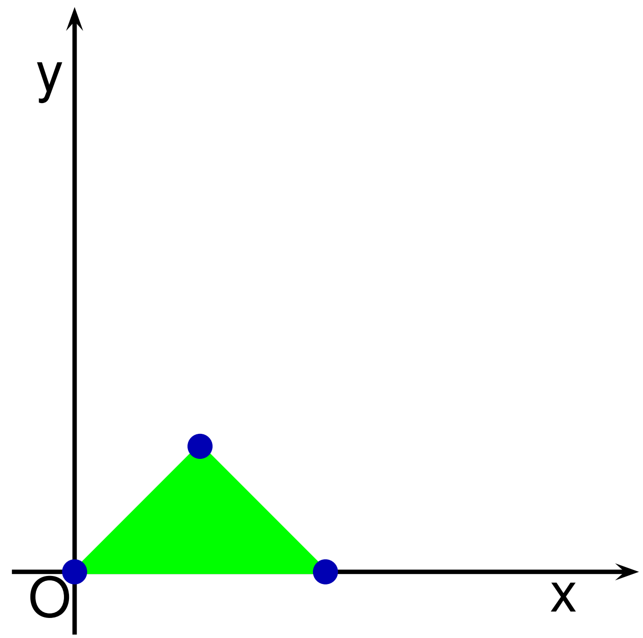
```
  else x := x+1; y := y+1;
```

$\{x = 1, y = 1\}$

```
  endif
```

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

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

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

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

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

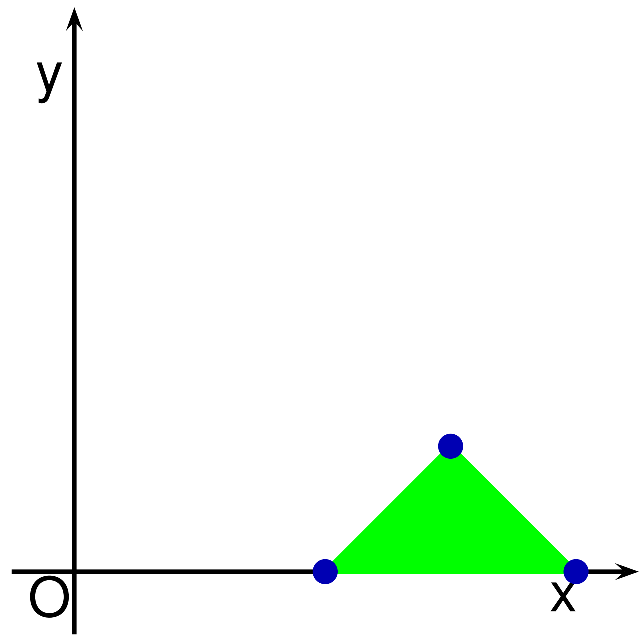
```
  else x := x+1; y := y+1;
```

$\{x = 1, y = 1\}$

```
  endif
```

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

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

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

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

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

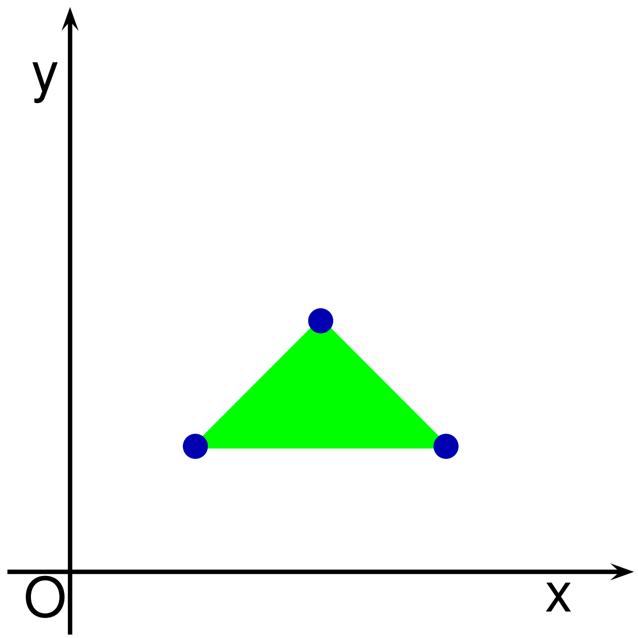
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

```
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;
```

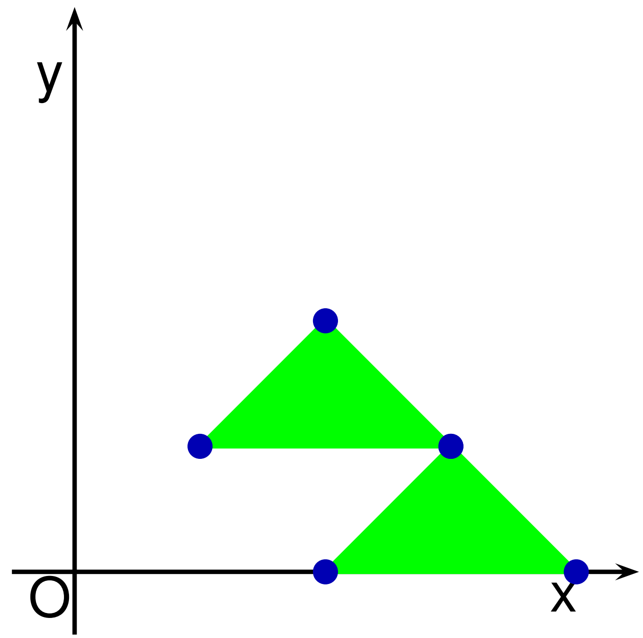
$\{1 \leq y \leq x, x + y \leq 4\}$

```
  endif
```

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

$\uplus \{1 \leq x \leq 2, x + y = 2\}$

```
endwhile
```



Example: The Abstract Semantics

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

$\{x = 0, y = 0\}$

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

$\{0 \le y \le x, x + y \le 2\}$

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

$\{0 \le y \le x - 2, x + y \le 4\}$

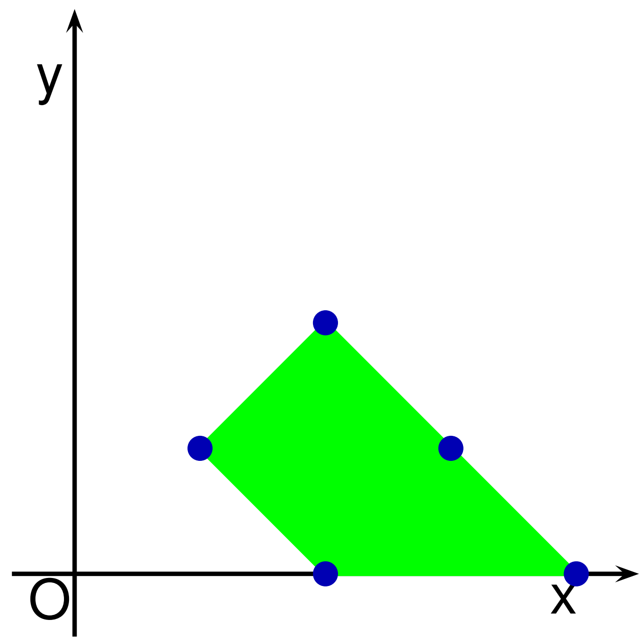
```
  else x := x+1; y := y+1;
```

$\{1 \le y \le x, x + y \le 4\}$

```
  endif
```

$\{0 \le y \le x, 2 \le x + y \le 4\}$

```
endwhile
```



EXAMPLE: ... AND SO ON ... ?

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x, x + y \leq 4\}$

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

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

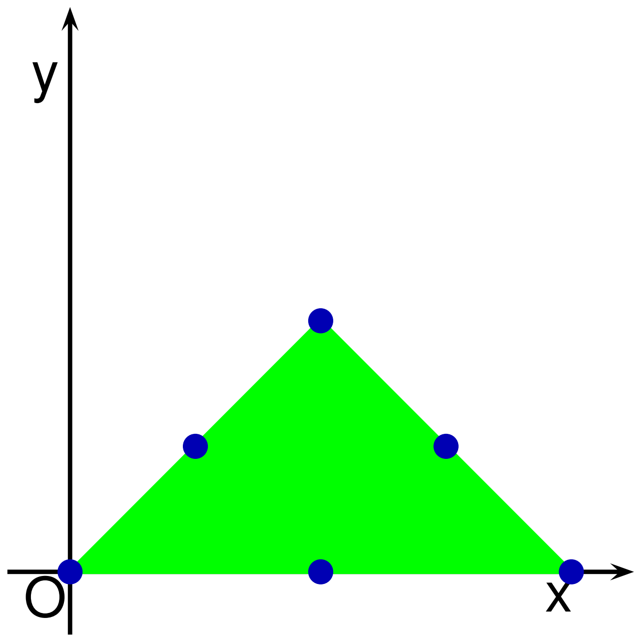
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: Finite Convergence using Widening

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

$\{x = 0, y = 0\}$

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

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

$\nabla \{0 \leq y \leq x, x + y \leq 4\}$

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

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

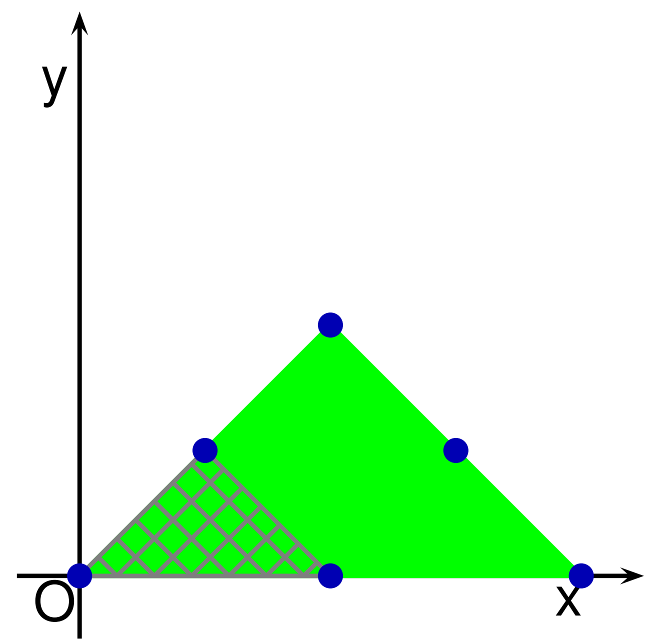
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: Finite Convergence using Widening

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

$\{x = 0, y = 0\}$

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

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

$\nabla \{0 \leq y \leq x, x + y \leq 4\}$

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

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

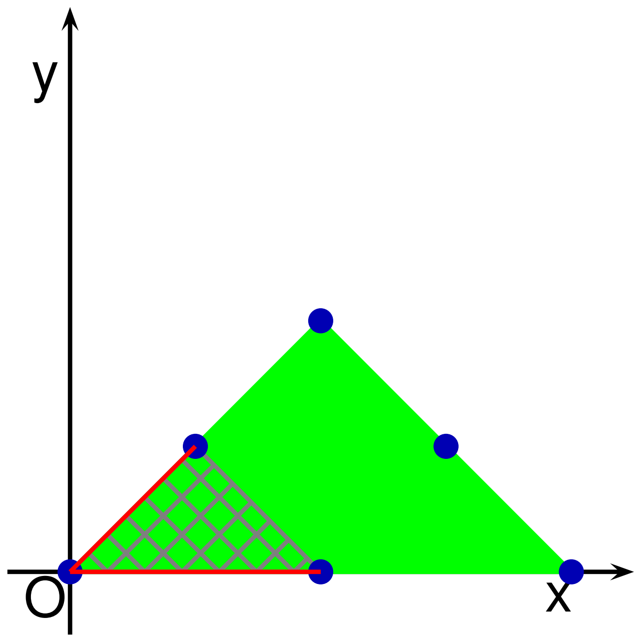
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: An Abstract Post-Fixpoint

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x\}$

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

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

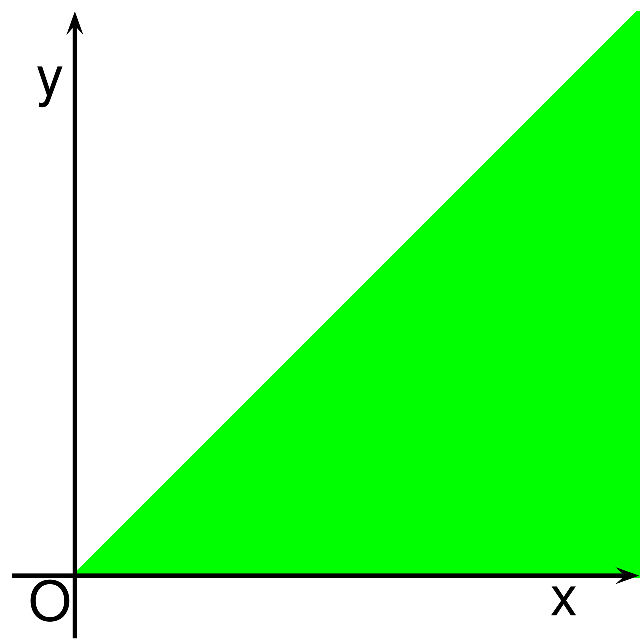
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: Abstract Downward Iteration

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x\}$

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

$\{0 \leq y \leq x - 2\}$

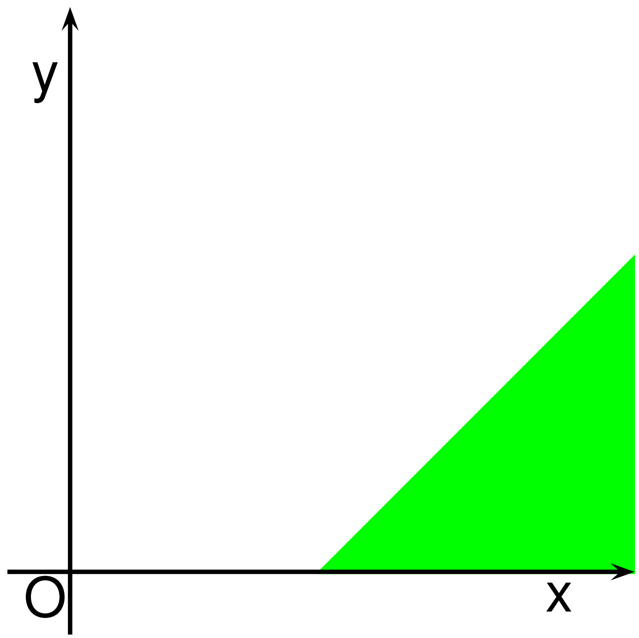
```
  else x := x+1; y := y+1;
```

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

```
  endif
```

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

```
endwhile
```



Example: Abstract Downward Iteration

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x\}$

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

$\{0 \leq y \leq x - 2\}$

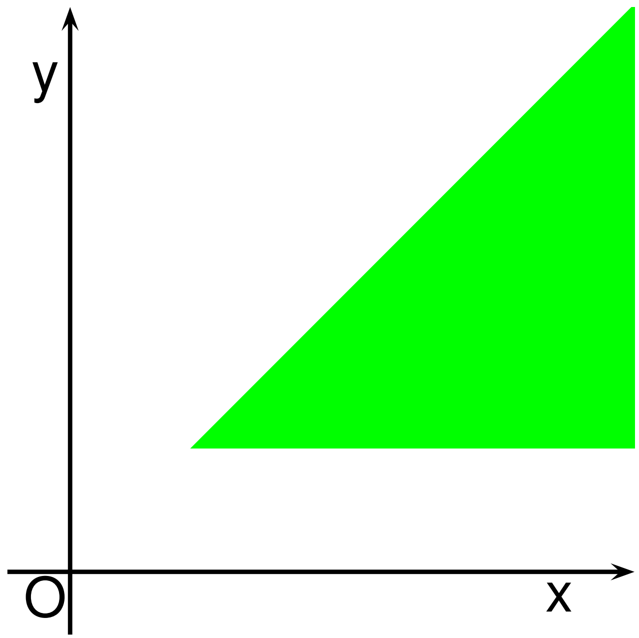
```
  else x := x+1; y := y+1;
```

$\{1 \leq y \leq x\}$

```
  endif
```

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

```
endwhile
```



Example: Abstract Downward Iteration

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x\}$

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

$\{0 \leq y \leq x - 2\}$

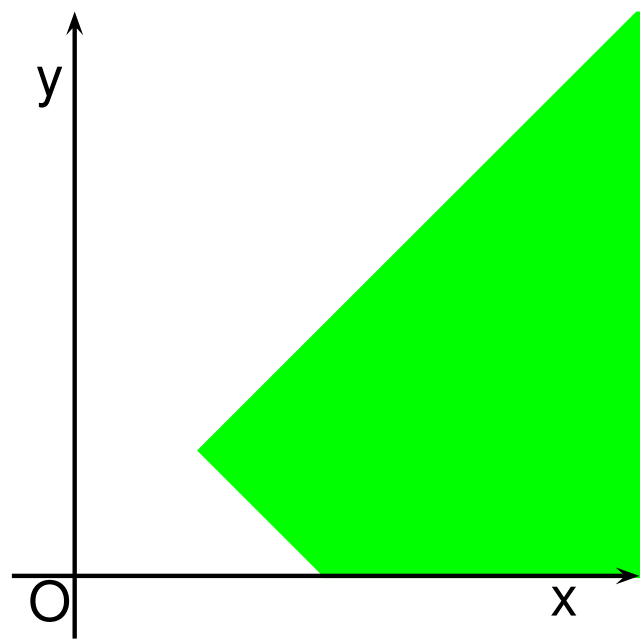
```
  else x := x+1; y := y+1;
```

$\{1 \leq y \leq x\}$

```
  endif
```

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

```
endwhile
```



Example: Abstract Fixpoint

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

$\{x = 0, y = 0\}$

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

$\{0 \leq y \leq x \leq 100\}$

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

$\{0 \leq y \leq x - 2\}$

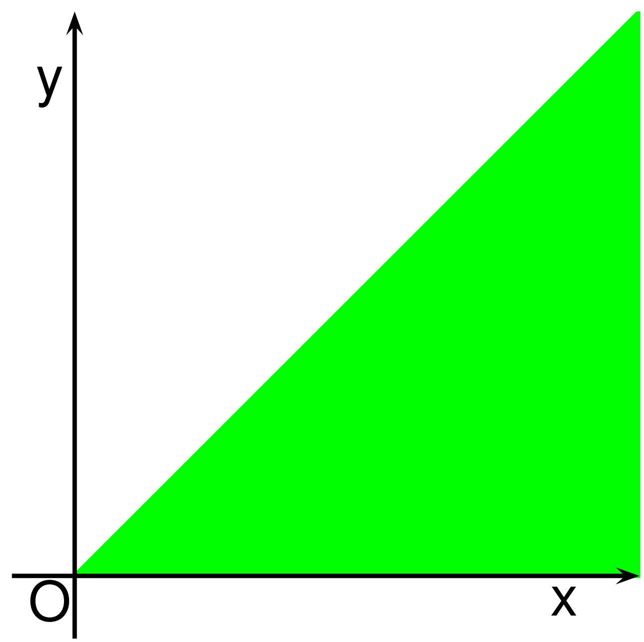
```
  else x := x+1; y := y+1;
```

$\{1 \leq y \leq x\}$

```
  endif
```

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

```
endwhile
```



Example: Abstract Fixpoint

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

$\{x = 0, y = 0\}$

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

$\{0 \le y \le x \le 100\}$

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

$\{0 \le y \le x - 2 \le 100\}$

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

$\{1 \le y \le x \le 101\}$

```
  endif
```

$\{0 \le y \le x \le 102, 2 \le x + y \le 202\}$

```
endwhile
```

$\{100 < x \le 102, 0 \le y \le x, x + y \le 202\}$

The (Well-Known) Moral of the Story

Semantic construction: language dependent, but (almost) independent from the specific application and, in particular, from the considered abstract domain.

Abstract Domain: semantic construction dependent, as far as the set of supported abstract operators is concerned. For some important cases (e.g., numerical abstractions) it is almost language and application independent.

For a better understanding of both theoretical and practical research issues, the above separation of concerns should be pursued as much as possible.

This talk is about abstract domains.

THE DOMAIN $\mathbb{CP}_n$ OF CLOSED CONVEX POLYHEDRA

A lattice $\langle \mathbb{CP}_n, \subseteq, \varnothing, \mathbb{R}^n, \uplus, \cap \rangle$, with infinite chains.

Constraint Representation: $\mathcal{P} = \mathrm{con}(\mathcal{C})$

- ➜ $\mathcal{C}$ is a finite set of linear non-strict inequality (resp., equality) constraints.
- ➜ No redundant constraint $+$ max number of equalities $\implies$ minimal form.
- ➜ Weak notions of canonical form (e.g., by orthogonality).

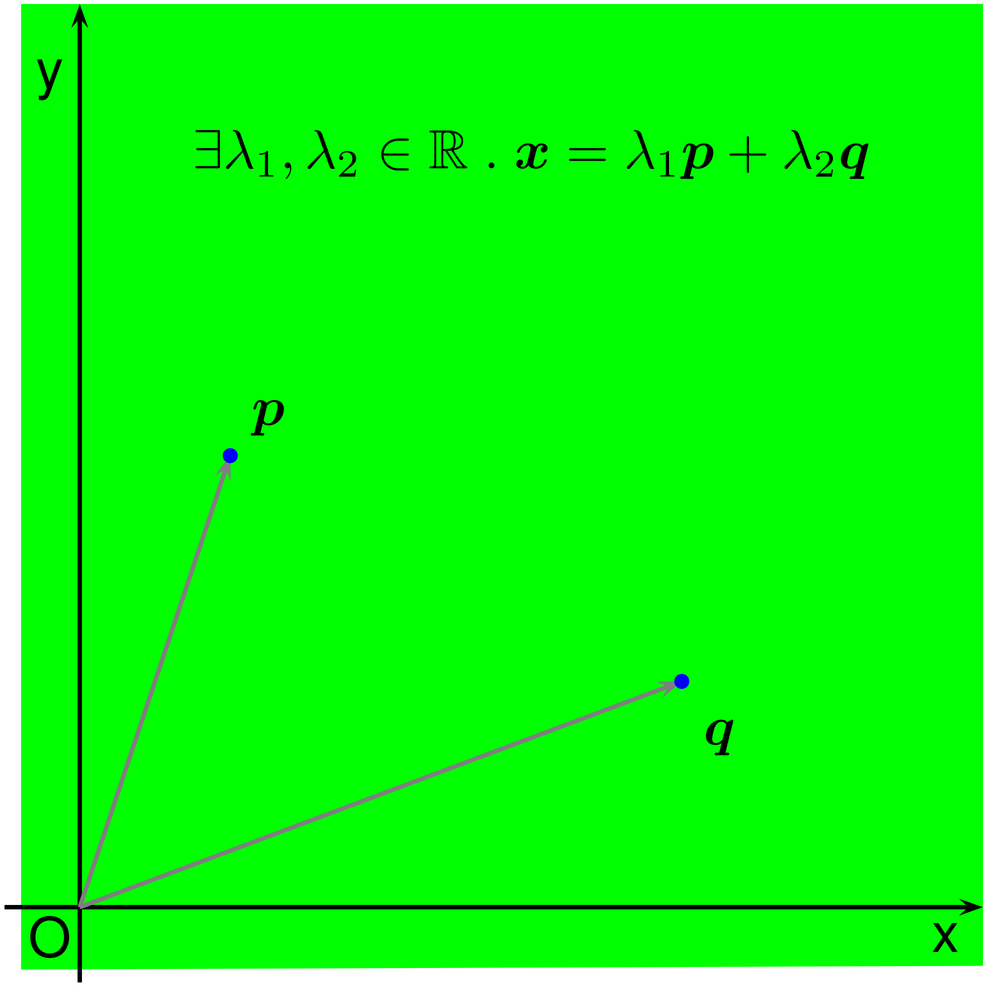
The Domain $\mathbb{CP}_n$ of Closed Convex Polyhedra

Generator Representation: $\mathcal{P} = \mathrm{gen}(\mathcal{G})$

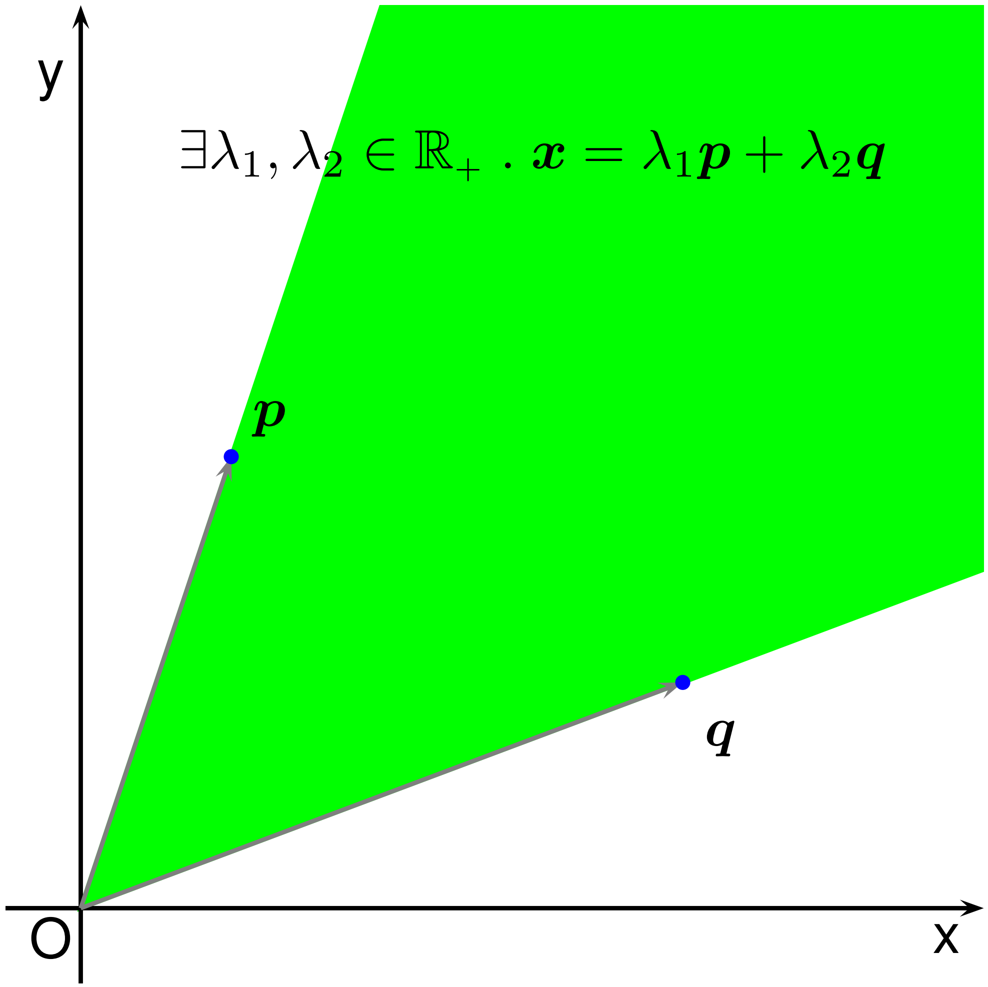
- $\mathcal{G} = (L, R, P)$, where
  - P is a finite set of points of $\mathcal{P}$;
  - R is a finite set of rays (directions of infinity) of $\mathcal{P}$;
  - L is a finite set of lines (bidirectional rays) of $\mathcal{P}$.
- No redundant generator $+$ max number of lines $\implies$ minimal form.
- Weak notions of canonical form (e.g., by orthogonality).

$$\mathrm{gen}(\mathcal{G}) \stackrel{\mathrm{def}}{=} \left\{ L\boldsymbol{\lambda} + R\boldsymbol{\rho} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{\lambda} \in \mathbb{R}^\ell, \boldsymbol{\rho} \in \mathbb{R}^r_+, \\ \boldsymbol{\pi} \in \mathbb{R}^p_+, \sum_{i=1}^p \pi_i = 1 \end{array} \right\}$$

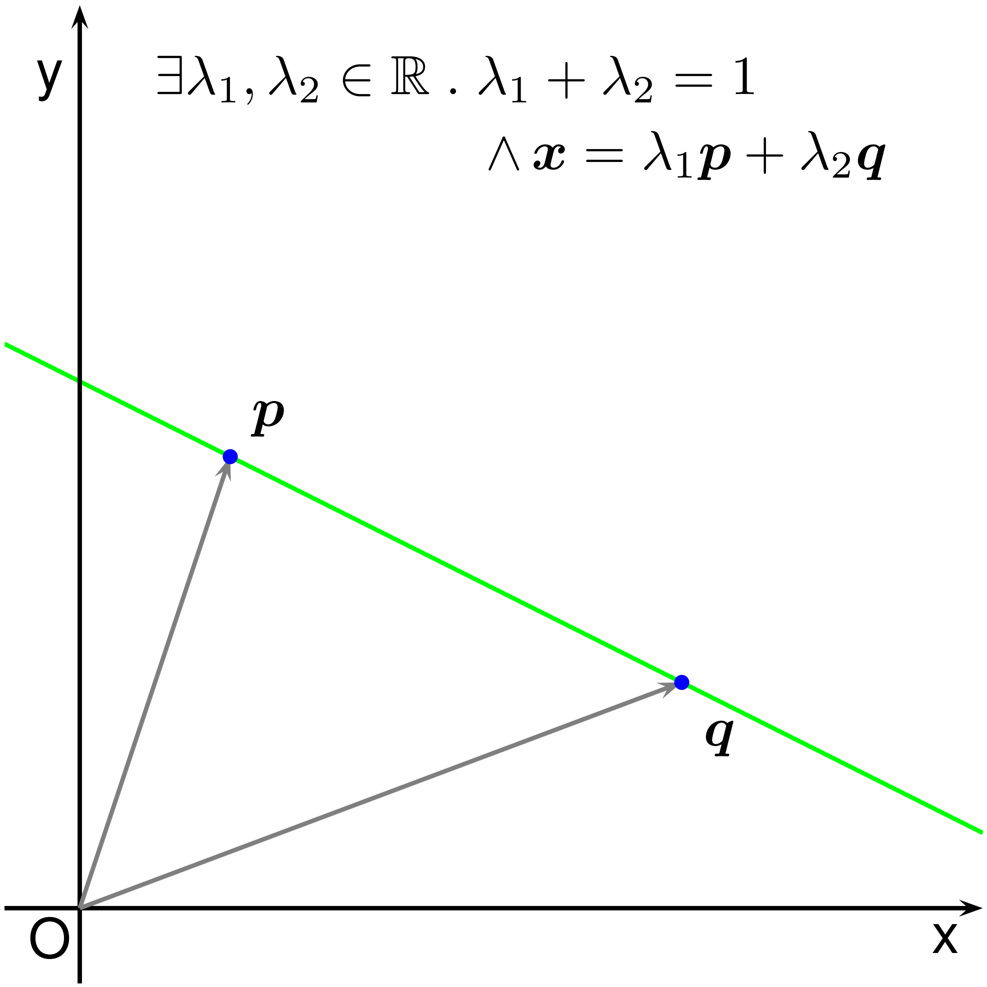
Combinations: Linear



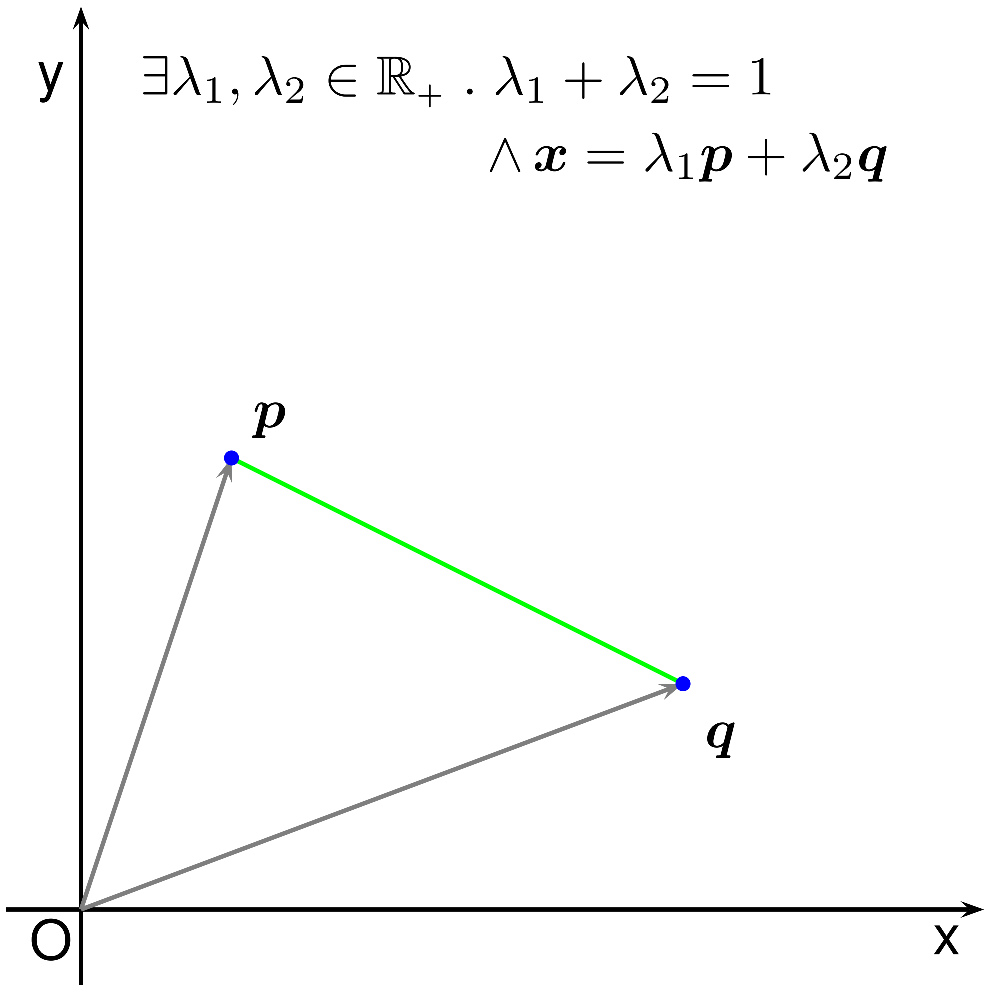
Combinations: Conic



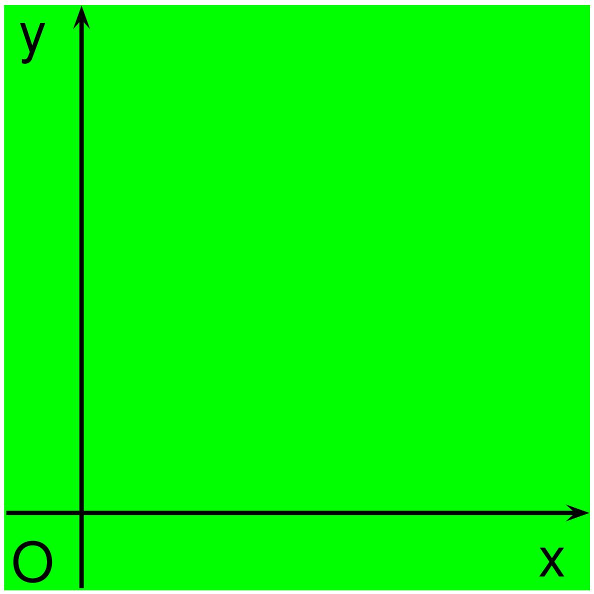
COMBINATIONS: AFFINE



$$\exists \lambda_1, \lambda_2 \in \mathbb{R}_+ \;.\; \lambda_1 + \lambda_2 = 1 \\ \wedge\, \boldsymbol{x} = \lambda_1 \boldsymbol{p} + \lambda_2 \boldsymbol{q}$$

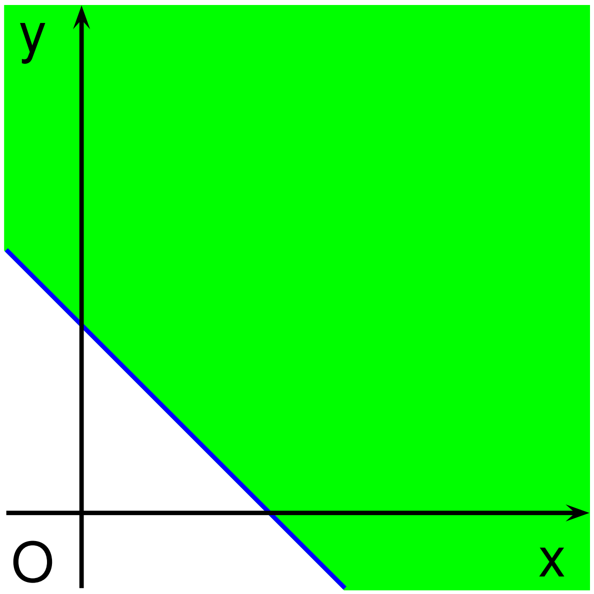


EXAMPLE: DOUBLE DESCRIPTION



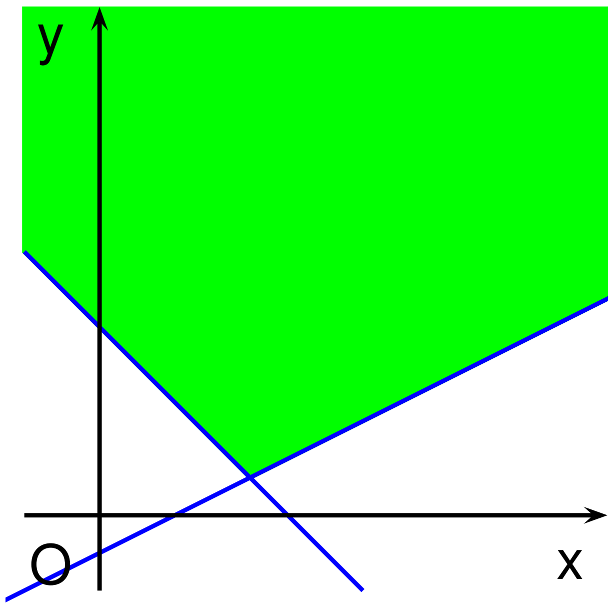
$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

Example: Double Description



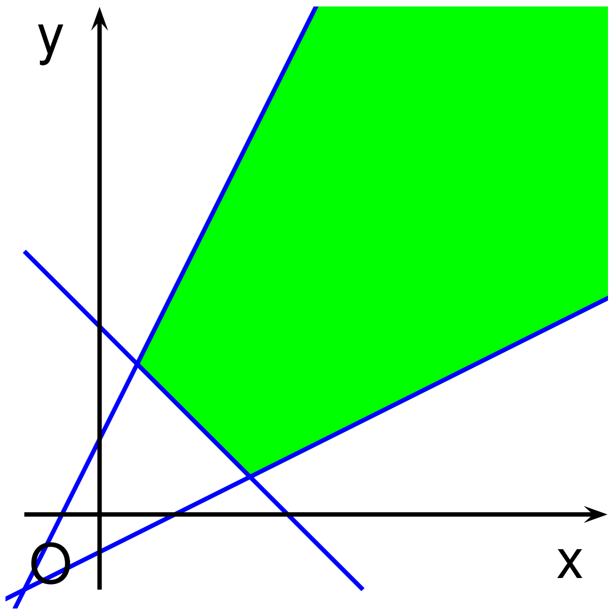
$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

Example: Double Description



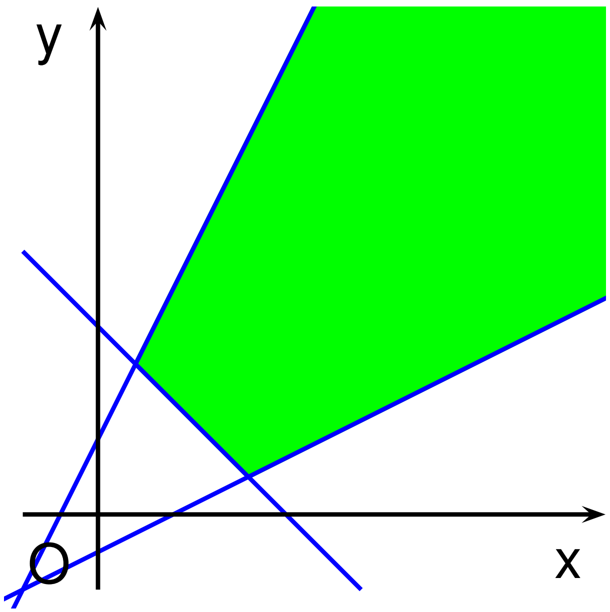
$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

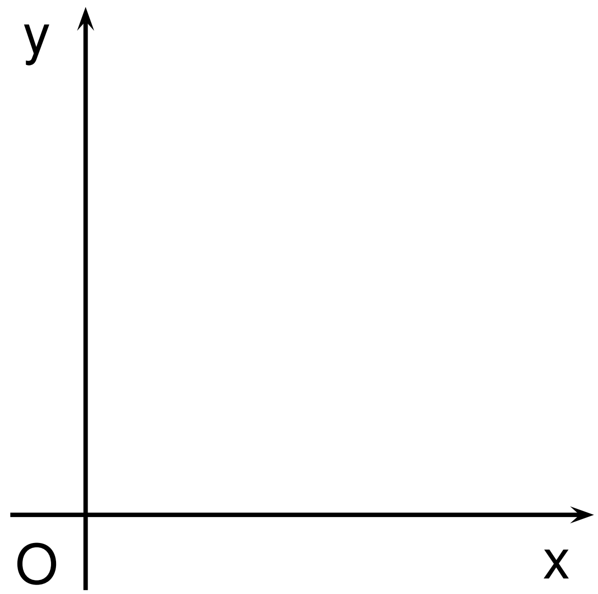
Example: Double Description



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

Example: Double Description

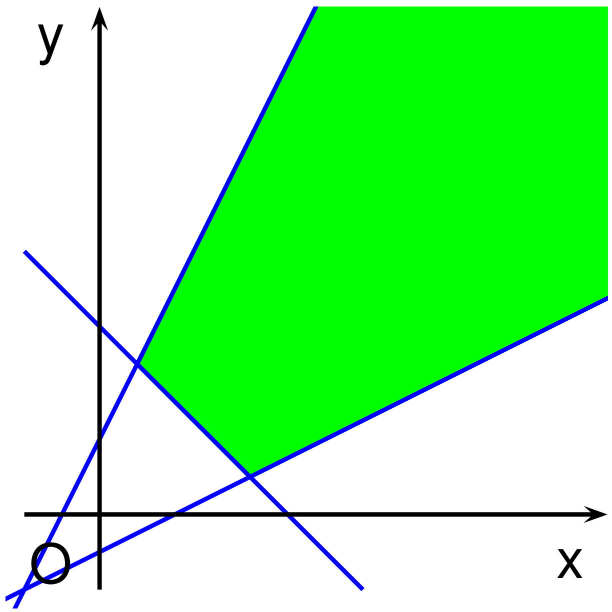


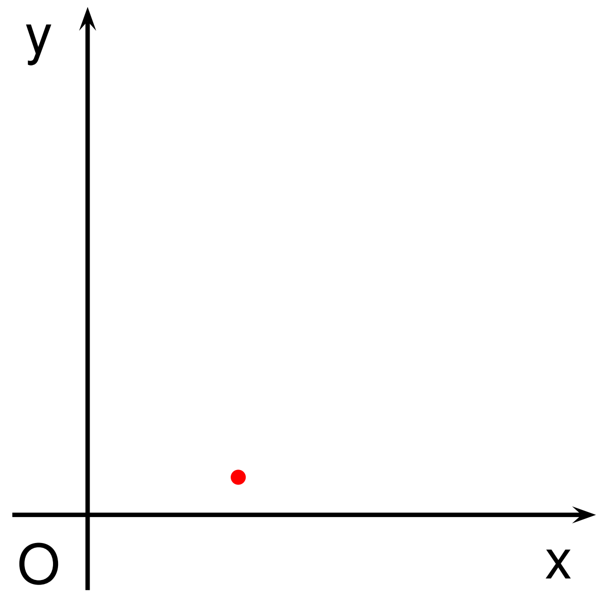


$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

$$\begin{cases} \text{lines: } \varnothing \\ \text{points: } \varnothing \\ \text{rays: } \varnothing \end{cases}$$

Example: Double Description

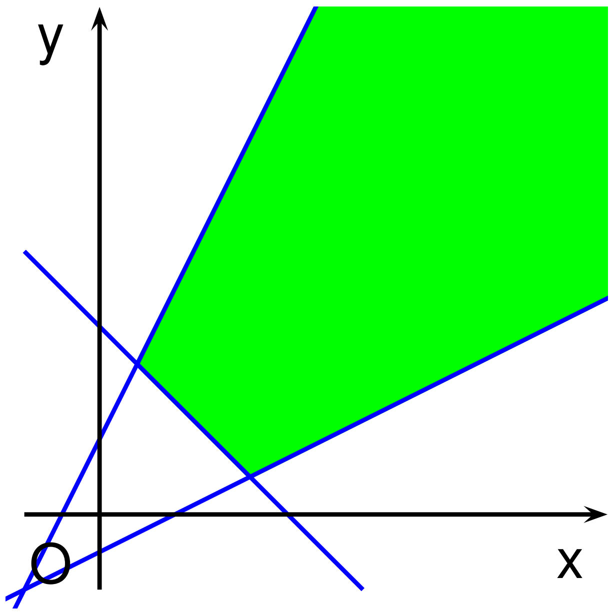


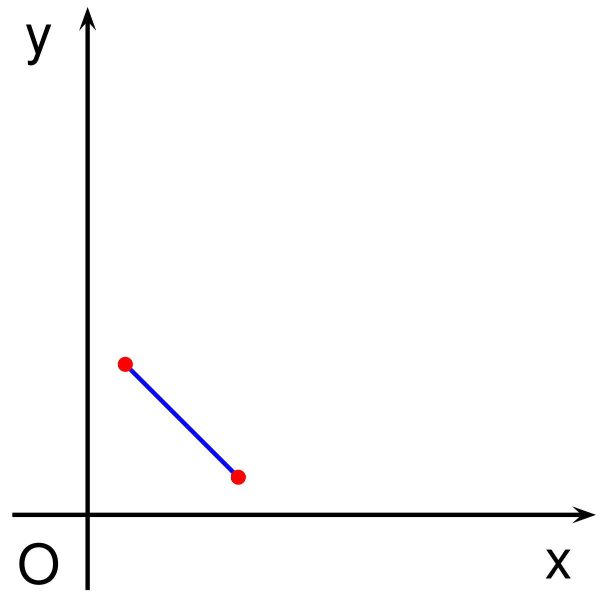


$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

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

Example: Double Description

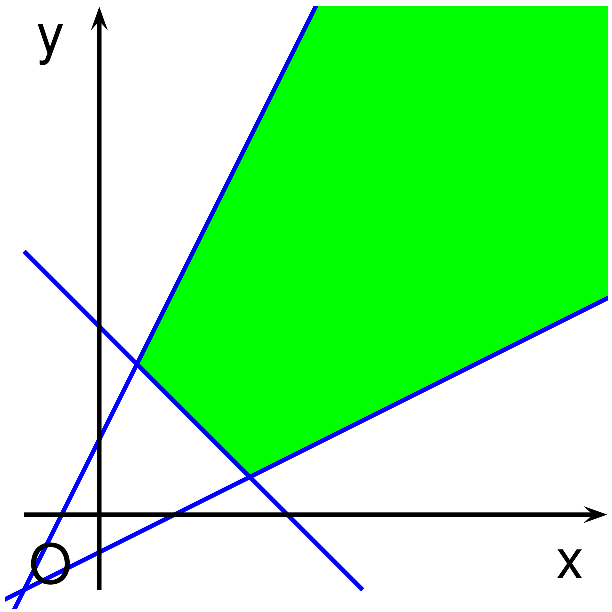


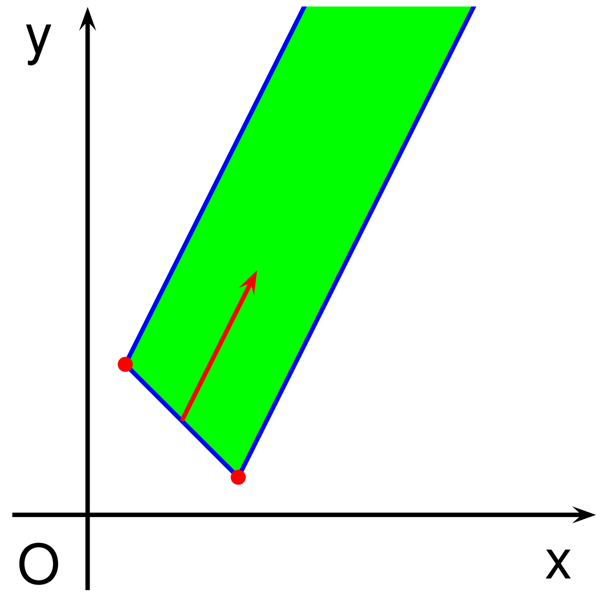


$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

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

Example: Double Description

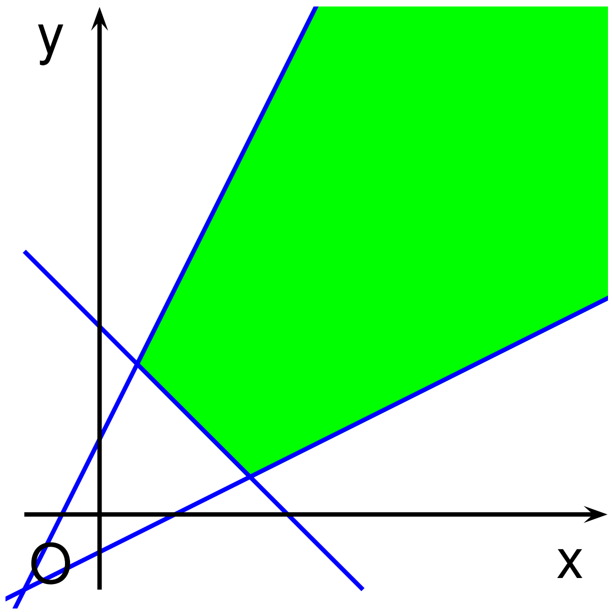


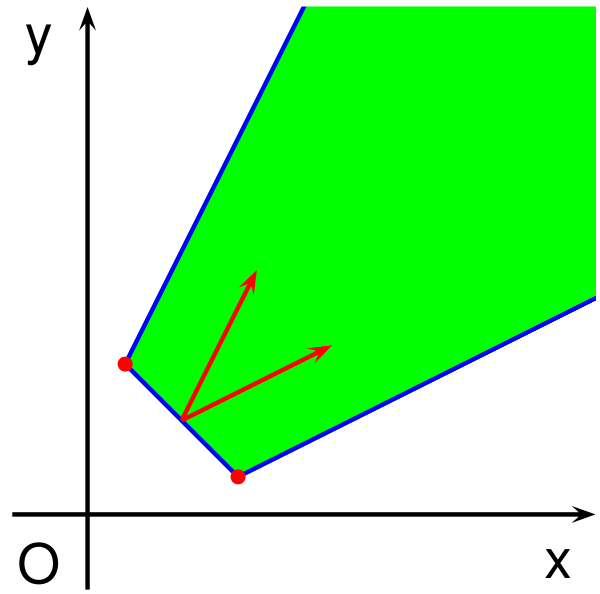


$$\begin{cases} x + y \geq 5 \\ x - 2y \leq 2 \\ y - 2x \leq 2 \end{cases}$$

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

Example: Double Description





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

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

THE DD METHOD BY MOTZKIN ET AL.

The Principle of Duality

- ➜ Systems of constraints and generators enjoy a 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;

$\Longrightarrow$ 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 DD pair.

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.

Example: The Inclusion Test

- Let $\mathcal{P}_1 = \mathrm{gen}(\mathcal{G}_1) \in \mathbb{CP}_n$ and $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2) \in \mathbb{CP}_n$.
- $\mathcal{P}_1 \subseteq \mathcal{P}_2$ iff each generator in $\mathcal{G}_1$ satisfies each constraint in $\mathcal{C}_2$;
  - generator $\boldsymbol{g} \in \mathbb{R}^n$ satisfies constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ if and only if the scalar product $s \stackrel{\mathrm{def}}{=} \langle \boldsymbol{a}, \boldsymbol{g} \rangle$ satisfies the following condition:

	Constraint type	
Generator type	$=$	$\geq$
line	$s = 0$	$s = 0$
ray	$s = 0$	$s \geq 0$
point	$s = b$	$s \geq b$

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.
  - Currently provides support for (not necessarily closed) convex polyhedra and finite sets of (NNC) polyhedra.
- 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 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* 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;
- 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);
  }
}
```

(KNOWN) USERS OF THE LIBRARY

- ➜ VERIMAG, FR (D. Merchat et al., Cartesian factoring)
- ➜ U. of Réunion, FR (F. Menard et al., cTI)
- ➜ Carnegie Mellon U., USA (K. Mixter et al., Action Language Verifier and G. Frehse, Linear Hybrid Automata)
- ➜ Delft U. of Technology, DK (M. Rhode)
- ➜ U. of Kent at Canterbury, UK (A. Simon, floating-point computations)
- ➜ ENS Cachan, FR (E. Fersman, model checking of hybrid systems)
- ➜ U. of Michigan, USA (H. Song, extending Spin to hybrid contexts)
- ➜ U. of Wisconsin, USA (D. Gopan et al., extending TVLA)
- ➜ Standford U., USA (S. Sankaranarayanan et al., StInG)
- ➜ U. of Cambridge, UK (E. Upton et al., gated data dependence graphs)
- ➜ U. of Tel Aviv, IL (M. Sagiv et al., string cleanness for C programs)
- ➜ ...

From Practice Back to Theory 1: Widenings

① The "limit" of the approximated computation may not be representable in the abstract domain (e.g., a circle is not a polyhedron);

② Reaching a post-fixpoint may still require an infinite number of computation steps;

③ Even when the computation is intrinsically finite, as was the case in the example we have seen, it may be practically unfeasible if it requires too many approximated iterations.

Widening operators try to solve all of these problems at once.

Definition of Widening Operator

A variant of the classical one (see Cousot and Cousot, PLILP'92):

- ➜ Let $\langle L, \sqsubseteq, \bot, \sqcup \rangle$ be a join-semi-lattice. Then, the operator $\nabla \colon L \times L \rightarrowtail L$ is a widening on L if
  ① $\forall x, y \in L : x \sqsubseteq y \implies y \sqsubseteq x \nabla y$;
  ② for all increasing chains $y_0 \sqsubseteq y_1 \sqsubseteq \cdots$, the chain defined by $x_0 \stackrel{\text{def}}{=} y_0, \ldots, x_{i+1} \stackrel{\text{def}}{=} x_i \nabla (x_i \sqcup y_{i+1}), \ldots$ is not strictly increasing.
- ➜ The upward iteration sequence with widenings (starting from $x_0 = \bot$)

$$x_{i+1} = \begin{cases} x_i, & \text{if } \mathcal{F}^\sharp(x_i) \sqsubseteq x_i; \\ x_i \nabla \big(x_i \sqcup \mathcal{F}^\sharp(x_i)\big), & \text{otherwise;} \end{cases}$$

converges (to a post-fixpoint of $\mathcal{F}^\sharp$) after a finite number of iterations.

∇-COMPATIBLE LIMITED GROWTH ORDERING

➜ Any widening ∇ induces on L a partial order relation $\sqsubseteq_\nabla$ satisfying the ascending chain condition (ACC); this is the reflexive and transitive closure of

$$\{ (x, z) \in L \times L \mid \exists y \in L \,.\, x \sqsubset y \wedge z = x \nabla y \}.$$

∇-COMPATIBLE LIMITED GROWTH ORDERING

- Any widening ∇ induces on L a partial order relation $\sqsubseteq_\nabla$ satisfying the ascending chain condition (ACC); this is the reflexive and transitive closure of

$$\{\, (x, z) \in L \times L \mid \exists y \in L \,.\, x \sqsubset y \wedge z = x \nabla y \,\}.$$

- A limited growth ordering (lgo) is the strict version of a finitely computable preorder relation that satisfies the ACC on L.
- Let ∇ be a widening on L. An lgo $\curvearrowright$ is ∇-compatible if

$$\forall x, y \in L : x \sqsubset y \implies x \curvearrowright x \nabla y.$$

∇-COMPATIBLE LIMITED GROWTH ORDERING

- Any widening ∇ induces on L a partial order relation $\sqsubseteq_\nabla$ satisfying the ascending chain condition (ACC); this is the reflexive and transitive closure of

$$\left\{ (x, z) \in L \times L \;\middle|\; \exists y \in L \,.\, x \sqsubset y \wedge z = x \nabla y \right\}.$$

- A limited growth ordering (lgo) is the strict version of a finitely computable preorder relation that satisfies the ACC on L.
- Let ∇ be a widening on L. An lgo $\curvearrowright$ is ∇-compatible if

$$\forall x, y \in L : x \sqsubset y \implies x \curvearrowright x \nabla y.$$

- A ∇-compatible lgo is a finite convergence certificate for ∇.

A Framework for Improving Upon a Fixed Widening

Suppose that

① $\nabla \colon L \times L \rightarrowtail L$ is a widening on the join-semi-lattice $\langle L, \sqsubseteq, \bot, \sqcup \rangle$;

② $\curvearrowright \subseteq L \times L$ is a ∇-compatible lgo;

③ $h \colon L \times L \rightarrow L$ is an upper bound operator.

For all $x, y \in L$ such that $x \sqsubseteq y$, define

$$x \mathbin{\tilde{\nabla}} y \stackrel{\text{def}}{=} \begin{cases} h(x,y), & \text{if } x \curvearrowright h(x,y) \sqsubset x \mathbin{\nabla} y; \\ x \mathbin{\nabla} y, & \text{otherwise.} \end{cases}$$

A Framework for Improving Upon a Fixed Widening

Suppose that

① $\nabla \colon L \times L \rightarrowtail L$ is a widening on the join-semi-lattice $\langle L, \sqsubseteq, \bot, \sqcup \rangle$;

② $\curvearrowright \subseteq L \times L$ is a ∇-compatible lgo;

③ $h \colon L \times L \to L$ is an upper bound operator.

For all $x, y \in L$ such that $x \sqsubseteq y$, define

$$x \mathbin{\tilde{\nabla}} y \stackrel{\mathrm{def}}{=} \begin{cases} h(x, y), & \text{if } x \curvearrowright h(x, y) \sqsubset x \mathbin{\nabla} y; \\ x \mathbin{\nabla} y, & \text{otherwise.} \end{cases}$$

➜ Then $\tilde{\nabla}$ is a widening operator at least as precise as ∇.

A New Widening for Convex Polyhedra

- In [Bagnara et al., SAS'03] we have instantiated the framework on the domain $\mathbb{CP}_n$, improving upon the standard widening.
- We have defined a fine-grained, standard widening-compatible lgo and used four different heuristics: do not widen, combining constraints, evolving points, evolving rays.
- Experiments have shown that the new widening significantly improves upon the precision of the standard widening.
- In general, this does not hold for the final result of upward iteration sequences, because neither the standard widening nor the new one are monotonic operators.
- Depending on the considered application, efficiency can be degraded. Several trade-off's are possible.

From Practice Back to Theory 2: Powerset Domains

- For the purposes of several applications, any convex set approximation is going to be too coarse. In these cases, the abstract domain should be enhanced to manipulate irregular geometric shapes.
- Often, the disjunction of a small number of convex approximations is enough to provide the required level of precision.

From Practice Back to Theory 2: Powerset Domains

➜ For the purposes of several applications, any convex set approximation is going to be too coarse. In these cases, the abstract domain should be enhanced to manipulate irregular geometric shapes.

➜ Often, the disjunction of a small number of convex approximations is enough to provide the required level of precision.

➜ The finite powerset domain is 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, ...

➜ ...together with a specific instance of the construction on the domain of convex polyhedra.

POWERSET DOMAINS

➜ The theory underlying the powerset construction is (more or less) standard: disjunctive completion was defined in [Cousot and Cousot, POPL'79].

Powerset Domains

- ➜ The theory underlying the powerset construction is (more or less) standard: disjunctive completion was defined in [Cousot and Cousot, POPL'79].
- ➜ However, for both theory and practice, really few works have considered the definition of widening operators on these enhanced domains.
- ➜ Thus, practical experiences have been confined to more or less selected contexts (e.g., model checking), where the finite convergence guarantee may be given up.

Widening Powerset Domains

- ➜ In [Bagnara et al., VMCAI'04] we have studied the problem of specifying a proper widening operator on the powerset domain by lifting a widening operator defined on the base-level domain.
- ➜ We have proposed three different approaches:
  1. one is based on the cardinality of the set of abstract elements;
  2. one is based on a connector operator, that has to match each element in the new set with (at least) one element of the old set;
  3. one requires that the base-level widening comes with a finite convergence certificate.
- ➜ The PPL offers an implementation of the third, certificate-based widening for the powerset of convex polyhedra. This is the first widening operator defined on this domain.

From Practice Back to Theory 3: NNC Polyhedra

Strict Inequalities and NNC Polyhedra

- 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.

Encoding NNC Polyhedra as C Polyhedra

- call $\mathbb{P}_n$ and $\mathbb{CP}_n$ the sets of all NNC and closed polyhedra, respectively;
- each NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ can be embedded into a closed polyhedron $\mathcal{R} \in \mathbb{CP}_{n+1}$:
- the additional dimension of the vector space, usually labeled by the letter ϵ, encodes the topological closedness of each affine half-space in the constraint description for $\mathcal{P}$.

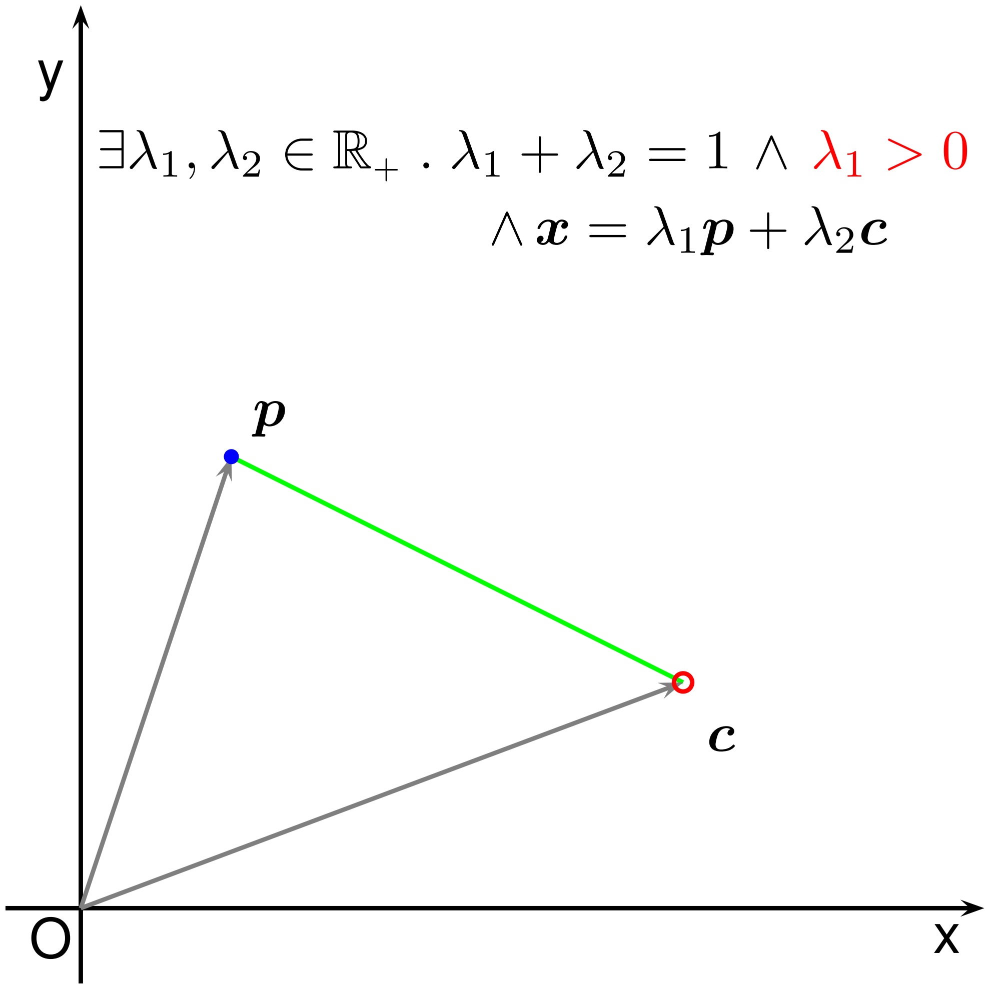
What Are the Generators of NNC Polyhedra?

- A fundamental feature of the DD method: the ability to represent polyhedra both by constraints and generators.
- Previous works/implementations did not offer a satisfactory answer.
- An intuitive generalization was provided in [Bagnara et al., SAS'02], based on the introduction of a new kind of generators: closure points.
- Extended generator systems: $\mathcal{G} = (L, R, P, C)$.

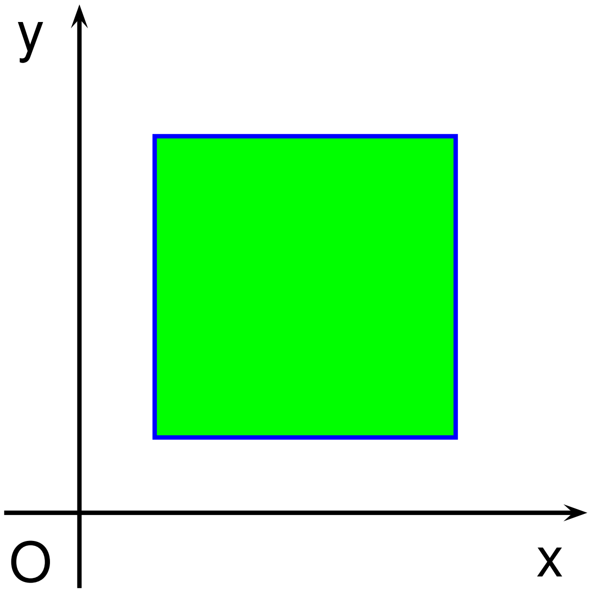
$$\operatorname{gen}(\mathcal{G}) \stackrel{\text{def}}{=} \left\{ L\boldsymbol{\lambda} + R\boldsymbol{\rho} + P\boldsymbol{\pi} + C\boldsymbol{\gamma} \;\middle|\; \begin{array}{l} \boldsymbol{\lambda} \in \mathbb{R}^{\ell}, \boldsymbol{\rho} \in \mathbb{R}^{r}_{+}, \boldsymbol{\pi} \in \mathbb{R}^{p}_{+}, \boldsymbol{\gamma} \in \mathbb{R}^{c}_{+}, \\ \sum_{i=1}^{p} \pi_i + \sum_{i=1}^{c} \gamma_i = 1, \boldsymbol{\pi} \neq \mathbf{0} \end{array} \right\}$$

Combinations: NNC Convex

$$\exists \lambda_1, \lambda_2 \in \mathbb{R}_+ \,.\, \lambda_1 + \lambda_2 = 1 \;\wedge\; \color{red}{\lambda_1 > 0} \\ \wedge\, \boldsymbol{x} = \lambda_1 \boldsymbol{p} + \lambda_2 \boldsymbol{c}$$

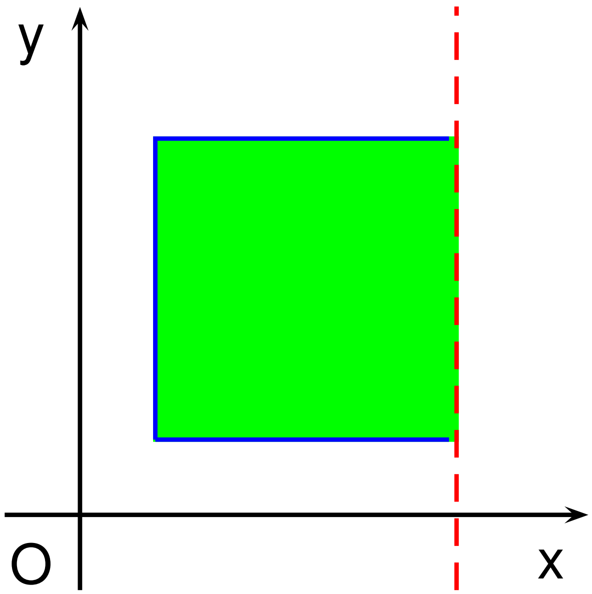


Example: NNC Double Description



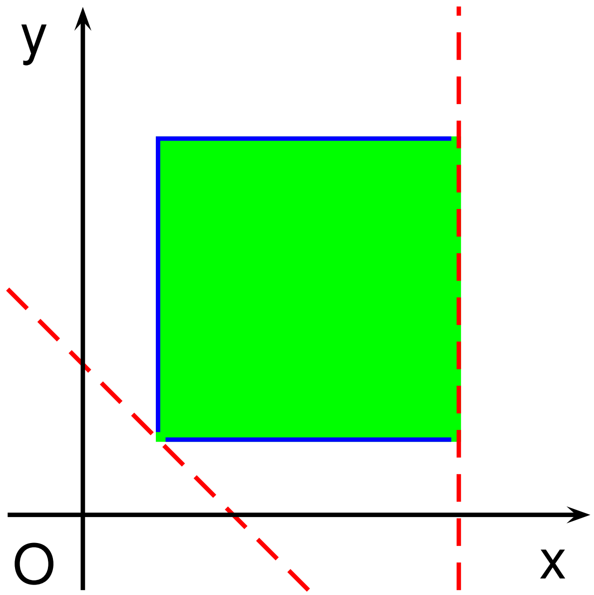
$$\begin{cases} 2 \leq x \leq 10 \\ 2 \leq y \leq 10 \\ \end{cases}$$

Example: NNC Double Description



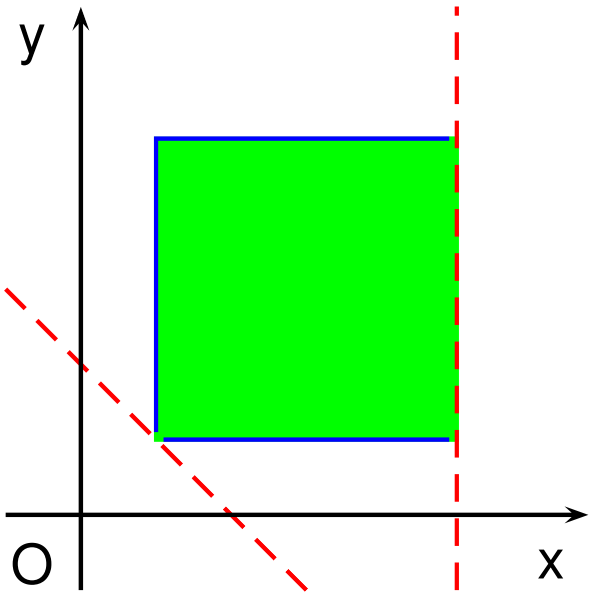
$$\begin{cases} 2 \leq x < 10 \\ 2 \leq y \leq 10 \\ \end{cases}$$

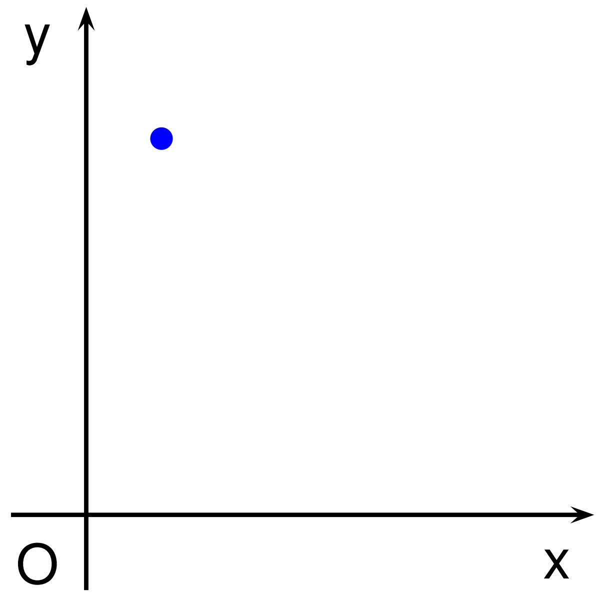
Example: NNC Double Description



$$\begin{cases} 2 \leq x < 10 \\ 2 \leq y \leq 10 \\ x + y > 4 \end{cases}$$

Example: NNC Double Description

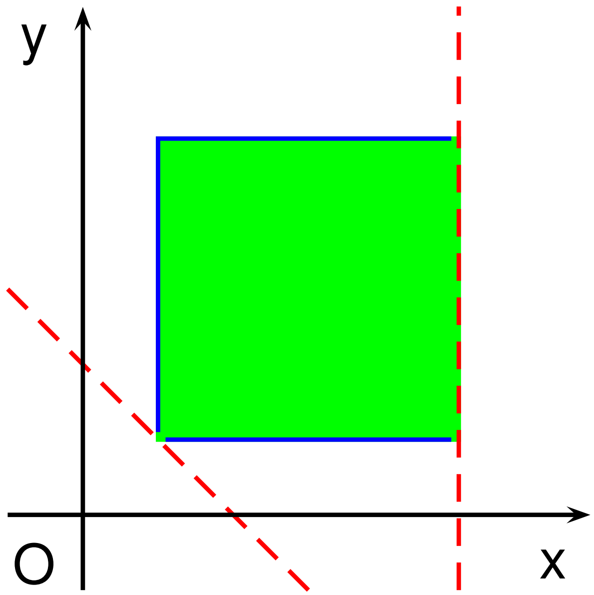


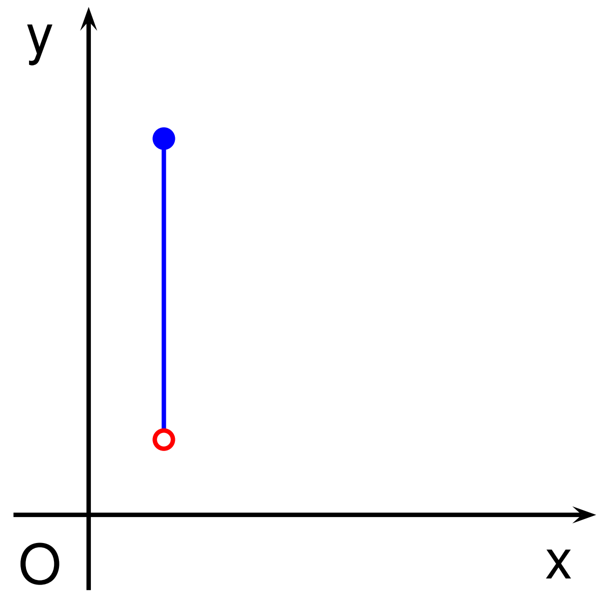


$$\begin{cases} 2 \leq x < 10 \\ 2 \leq y \leq 10 \\ x + y > 4 \end{cases}$$

$$\begin{cases} \text{lines: } \emptyset \qquad \text{rays: } \emptyset \\ \text{points: } \{(2, 10)\} \\ \text{c.p.: } \emptyset \end{cases}$$

Example: NNC Double Description

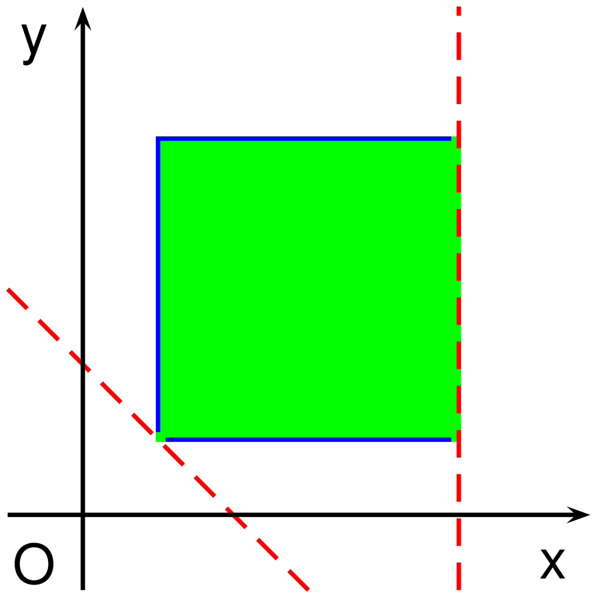


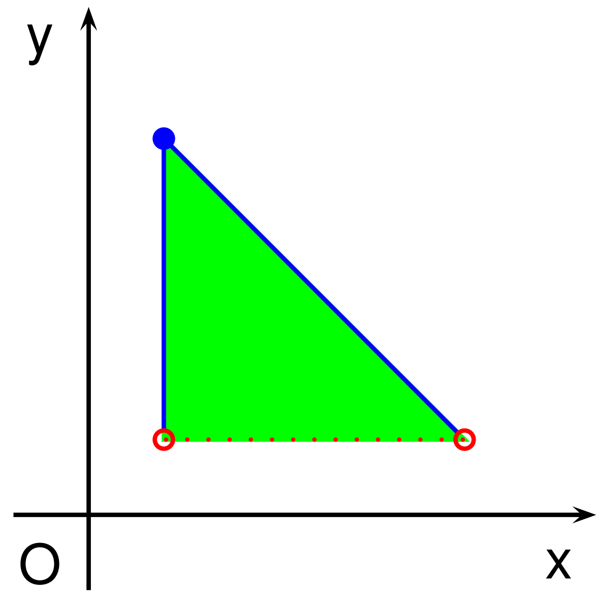


$$\begin{cases} 2 \leq x < 10 \\ 2 \leq y \leq 10 \\ x + y > 4 \end{cases}$$

$$\begin{cases} \text{lines: } \emptyset \qquad \text{rays: } \emptyset \\ \text{points: } \{(2, 10)\} \\ \text{c.p.: } \{(2, 2)\} \end{cases}$$

Example: NNC Double Description

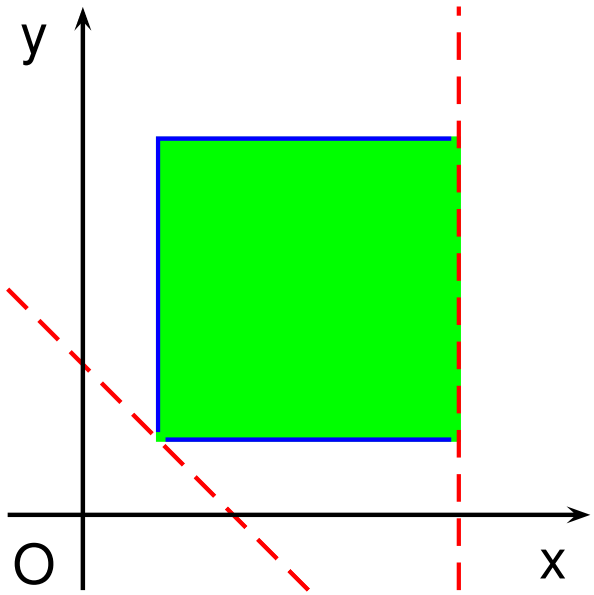


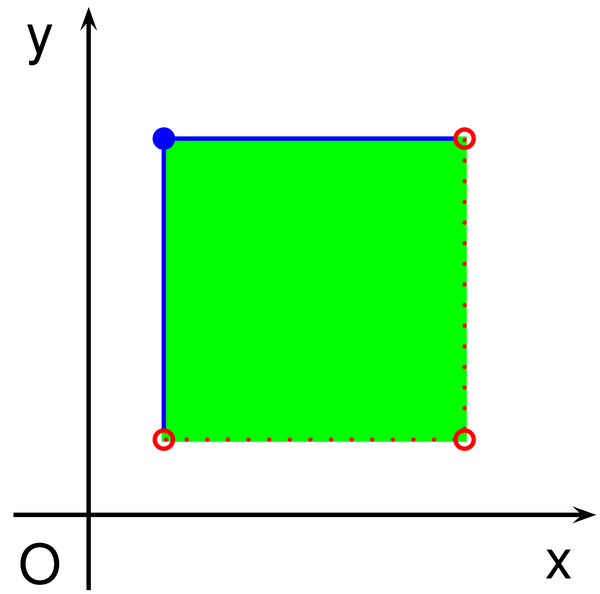


$$\begin{cases} 2 \leq x < 10 \\ 2 \leq y \leq 10 \\ x + y > 4 \end{cases}$$

$$\begin{cases} \text{lines: } \emptyset \qquad \text{rays: } \emptyset \\ \text{points: } \{(2, 10)\} \\ \text{c.p.: } \{(2, 2), (10, 2)\} \end{cases}$$

Example: NNC Double Description

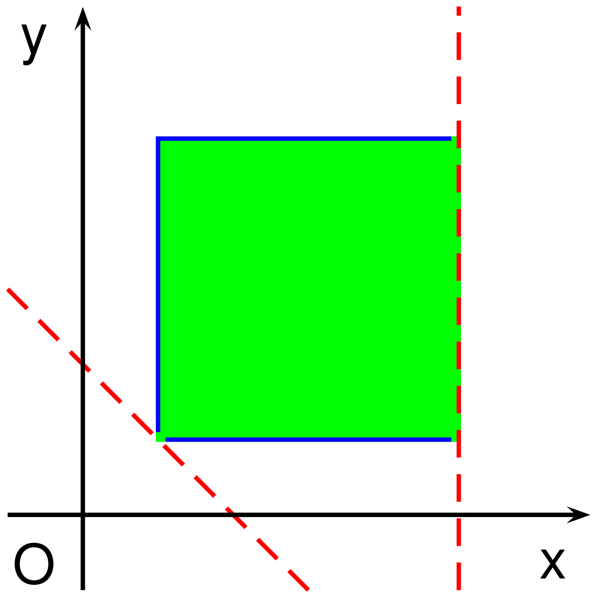


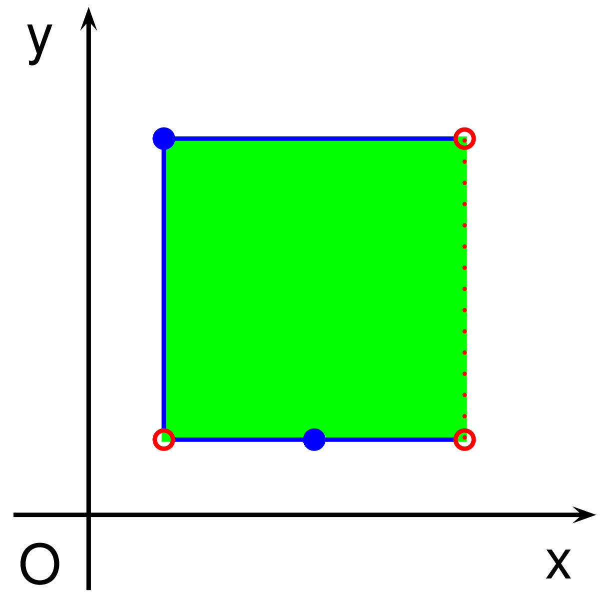


$$\begin{cases} 2 \le x < 10 \\ 2 \le y \le 10 \\ x + y > 4 \end{cases}$$

$$\begin{cases} \text{lines: } \emptyset \qquad \text{rays: } \emptyset \\ \text{points: } \{(2, 10)\} \\ \text{c.p.: } \{(2, 2), (10, 2), (10, 10)\} \end{cases}$$

EXAMPLE: NNC DOUBLE DESCRIPTION





$$\begin{cases} 2 \le x < 10 \\ 2 \le y \le 10 \\ x + y > 4 \end{cases}$$

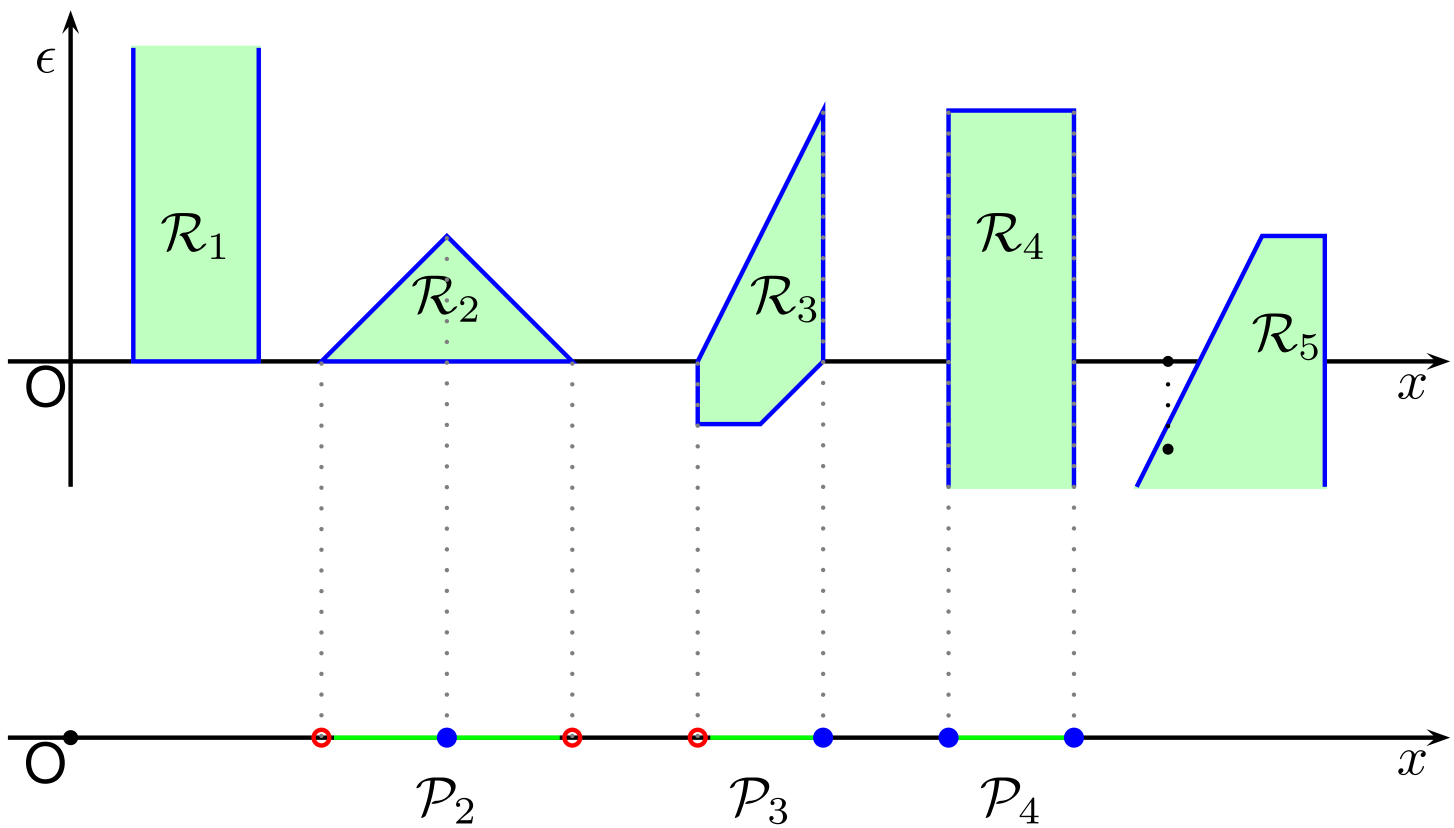
$$\begin{cases} \text{lines: } \emptyset \qquad \text{rays: } \emptyset \\ \text{points: } \{(2,10),(6,2)\} \\ \text{c.p.: } \{(2,2),(10,2),(10,10)\} \end{cases}$$

Example: The Inclusion Test for NNC Polyhedra

- Let $\mathcal{P}_1 = \mathrm{gen}(\mathcal{G}_1) \in \mathbb{P}_n$ and $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2) \in \mathbb{P}_n$.
- $\mathcal{P}_1 \subseteq \mathcal{P}_2$ iff each generator in $\mathcal{G}_1$ satisfies each constraint in $\mathcal{C}_2$.
- Generator $\boldsymbol{g} \in \mathbb{R}^n$ satisfies constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ if and only if the scalar product $s \stackrel{\text{def}}{=} \langle \boldsymbol{a}, \boldsymbol{g} \rangle$ satisfies the following condition:

	Constraint type		
Generator type	$=$	$\geq$	$>$
line	$s = 0$	$s = 0$	$s = 0$
ray	$s = 0$	$s \geq 0$	$s \geq 0$
point	$s = b$	$s \geq b$	$s > b$
closure point	$s = b$	$s \geq b$	$s \geq b$

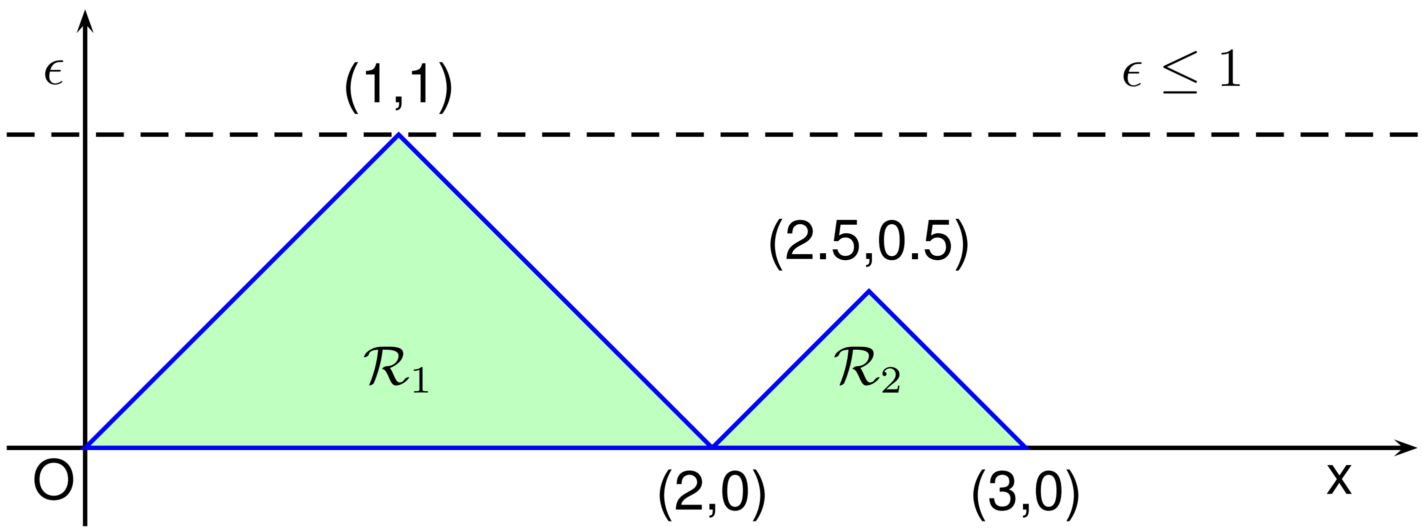
NNC IMPLEMENTATION: ϵ-POLYHEDRA



NNC Implementation: A Minimization Issue

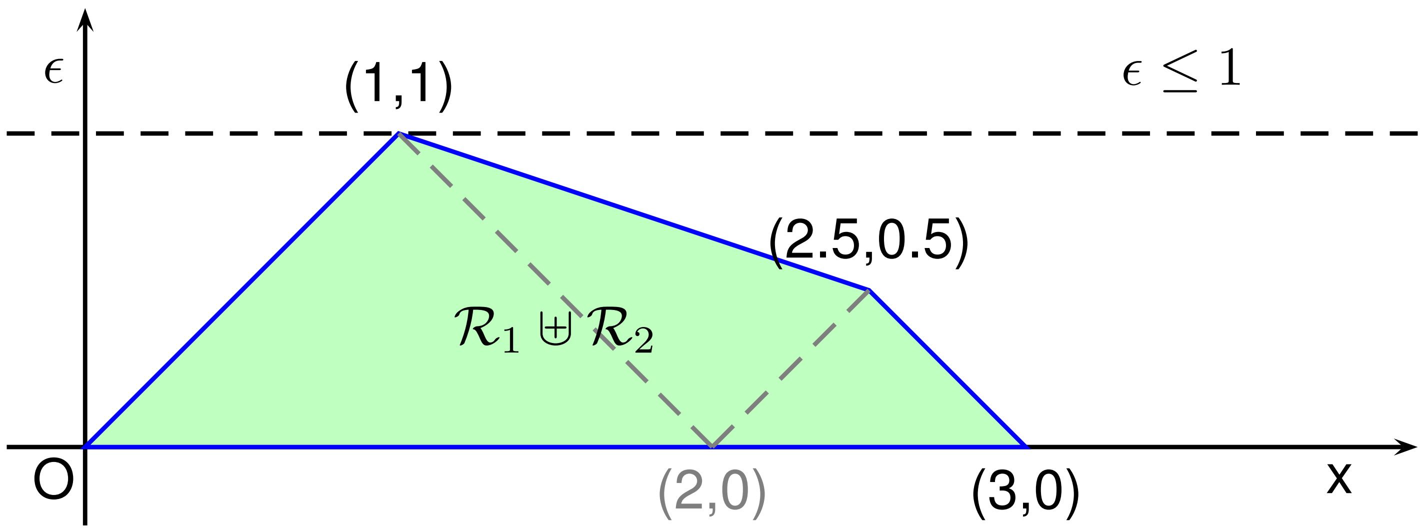
$\mathcal{R}_1$ encodes $\mathcal{P}_1 = \mathrm{con}\big(\{0 < x < 2\}\big)$,

$\mathcal{R}_2$ encodes $\mathcal{P}_2 = \mathrm{con}\big(\{2 < x < 3\}\big)$.

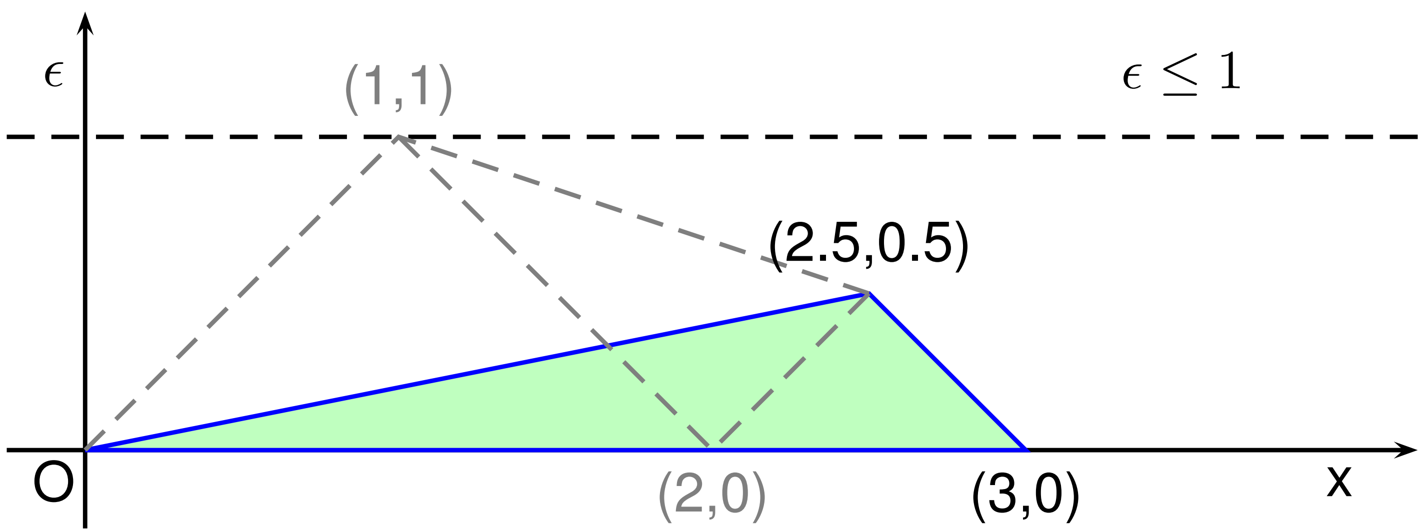


NNC Implementation: A Minimization Issue

$\mathcal{R}_1 \uplus \mathcal{R}_2$ encodes the poly-hull $\mathcal{P}_1 \uplus \mathcal{P}_2 = \mathrm{con}\big(\{0 < x < 3\}\big)$, but it also encodes the redundant constraint $x < 4$.



NNC IMPLEMENTATION: ϵ-MINIMAL FORMS!



Future PPL Features

Support for special classes of polyhedra

- An implementation of bounded differences and octagons.
- Work is in progress on a careful implementation of bounding boxes.
- Distinctive features are 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.

Finite Powersets of the above domains

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/`.