Applications of Polyhedral Computations to the Analysis and Verification of Hardware and Software Systems

Roberto Bagnara
Department of Mathematics
University of Parma
Italy
Joint work with Patricia M. Hill and Enea Zaffanella
(Department of Mathematics, University of Parma, Italy)

Part 1

How I Got Involved in All This

BACK IN 1992. . .

- ➜ . . . Giorgio invited me to look at the analysis of CLP over numeric domains: CLP($\mathcal{FD}$) and CLP($\mathcal{Q}$).
- ➜ I had read Cousot and Halbwachs, but developing an implemention of generic polyhedra sounded scary at the time (and, most importantly, did not fit the schedule for graduation).
- ➜ With Giorgio and Roberto Giacobazzi, we thus looked at constraint networks and propagation procedures upon them.
- ➜ These originated in the field of artificial intelligence in the ’80s and, as far as we could tell at the time, their use in static analysis and abstract interpretation had not been investigated.
  - Concerning static analysis we were wrong: see, e.g., Balasundaram and Kennedy 1989.
- ➜ Back then, we reformulated these techniques as abstract domains. Everything was implemented in my Prolog/CLP analyzer.

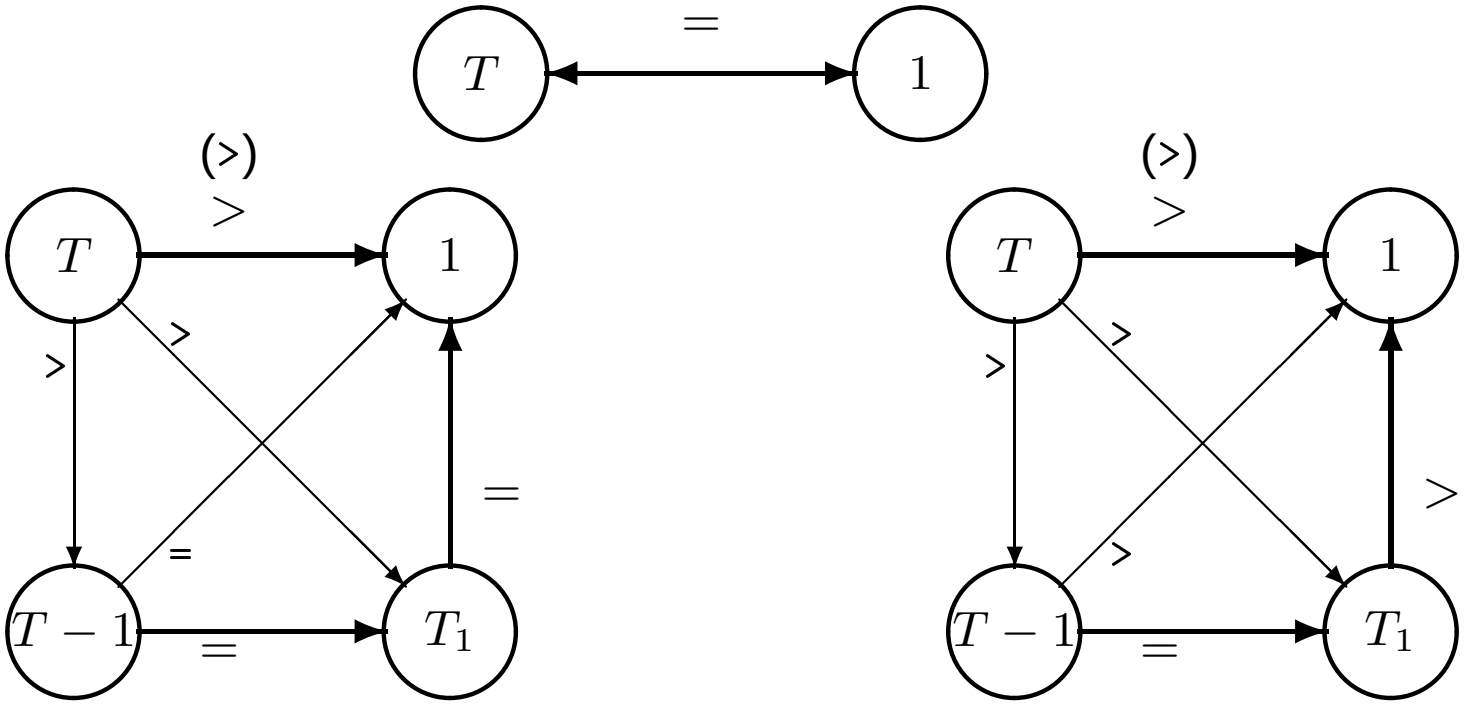
Before Polyhedra

- These domains were syntactic, i.e., the obtained result depended on the syntactic representation of constraints.
- Still, they were perfectly adequate for the detection of future redundant constraints:

$$
\begin{array}{lcl}
\texttt{mortgage}(P, T, I, R, B) & :- & T = 1, \\
 & & B = P * (1 + I/1200) - R \\
 & \square & . \\
\texttt{mortgage}(P, T, I, R, B) & :- & T > 1,\ T_1 = T - 1,\ P \geq 0, \\
 & & P_1 = P * (1 + I/1200) - R \\
 & \square & \texttt{mortgage}(P_1, T_1, I, R, B).
\end{array}
$$

Future Redundant Constraints

➜ In the abstract model of $\texttt{mortgage}/5$ there are:



➜ The last two networks show the existence of the entailed constraint $T > 1$, which does not depend on the textual constraint $T > 1$.

$\Longrightarrow$ $T > 1$ is future redundant!

Just a Taste of Polyhedra

- Sometime in the ’90s (probably 1994–1995), we obtained the Polka library (by Halbwachs et al.) in binary format.
- Unfortunately I could not use it, apparently due to some compiler incompatibility.
- But the few tests I could run made me curious.

- Nonetheless, I kept doing other things for several years:
  - I got interested into non-numerical properties of Prolog/CLP computation;
  - relatively few Prolog programs could benefit from sophisticated numerical domains.

- Until the day where I decided to look at other programming paradigms. . .

Part 2

Polyhedral Computation and Software/Hardware Verificati

Validation of Array References

Are these array accesses safe?

```
procedure shellsort(n : integer, array [0..n-1] of integer)
begin
  var h, i, j, B : integer;
  h := 1;
  while (h*3 + 1) < n do h := 3*h + 1;
  while h > 0 do
    i := h-1;
    while i < n do
      B := a[i]; j := i;
      while (j >= h) and (a[j-h] > B) do
        a[j] := a[j-h]; j := j-h;
      a[j] := B;
      i := i+1;
    h := h div 3;
```

String Cleanness in C/C++

Taken from Web2c: an implementation of TeX and friends that translates the original WEB sources into C. See, `http://www.tug.org/web2c/`.

```
#define BUFSIZ 1024
char buf[BUFSIZ];

char* insert_long(char *cp) {
  char temp[BUFSIZ];
  int i;
  assert(cp >= buf[0] && cp < buf[BUFSIZ]);
  for (i = 0; &buf[i] < cp; ++i)
    temp[i] = buf[i];
  strcpy (&temp[i], "(long)");              /* UNSAFE! */
  strcpy (&temp[i + 6], cp);                /* UNSAFE! */
  strcpy (buf, temp);
  return cp + 6;                            /* UNSAFE! */
}
```

Overflow of Signed Integers C/C++

Did you know that. . .

- . . . in C/C++ a signed integer overflow results in undefined behavior?
- Not to be confused with unspecified behavior.
- "Unspecified behavior" means that each combination of architecture, operating system and compiler must consistently define what happens on overflow:
  - saturation? wrap 2's (or 1's) complement? exception? termination?
- "Undefined behavior" means that anything can happen:
  - demons may fly out of your nose.
- Don't laugh please: compilers are increasingly exploiting undefined behavior in optimized compilation:
  - see the recent exploitable bug in Linux 2.6.30.

Example: The Concrete Semantics

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

while x <= 100 do
    (x, y) ∈ S ∈ ℘(ℝ²)
  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:

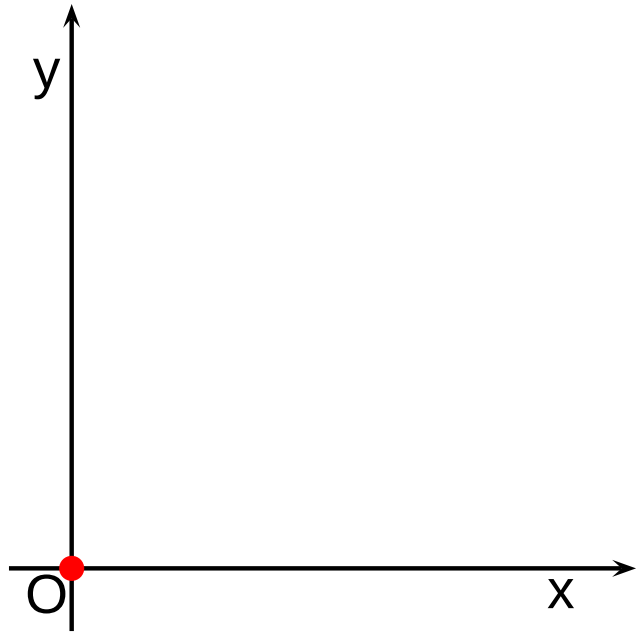
$$S \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F} = \mathcal{F}^{\omega}(\varnothing).$$

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
```

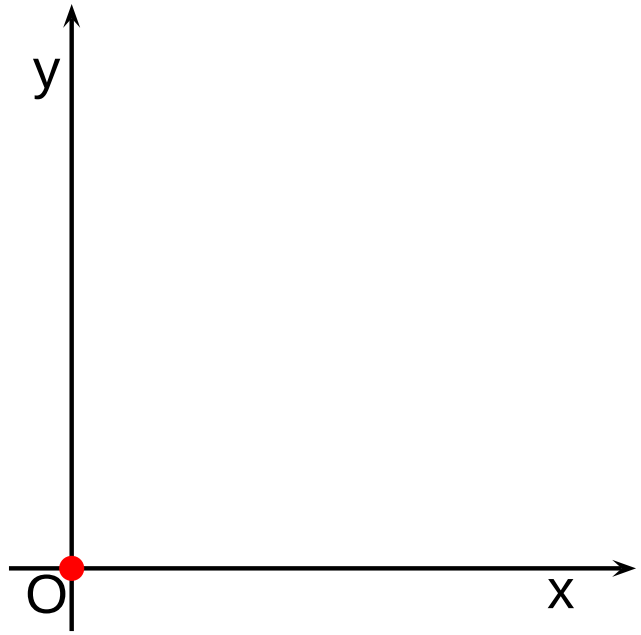
Example: The Concrete Semantics

```
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
```



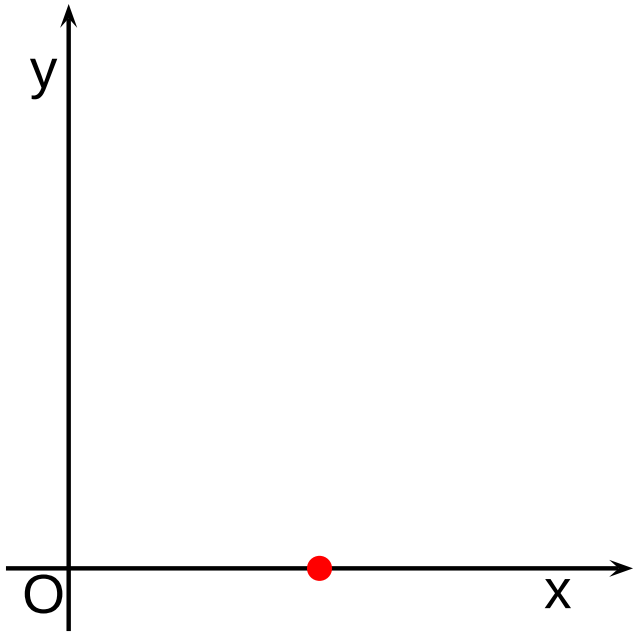
Example: The Concrete Semantics

```
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
```



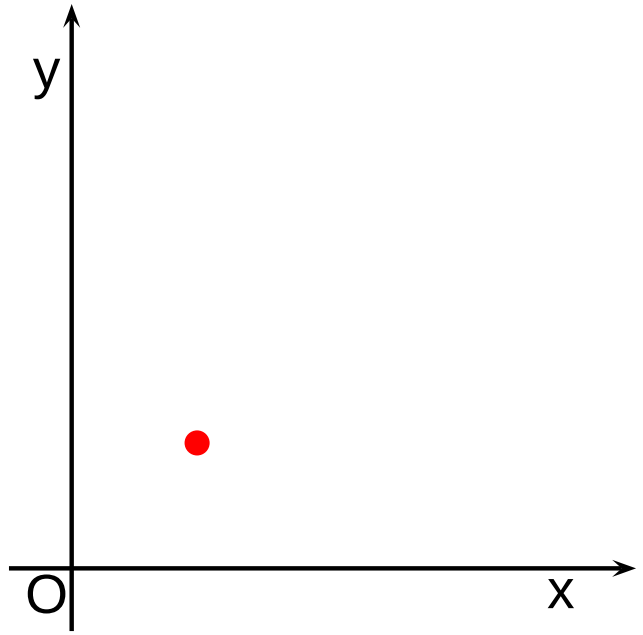
Example: The Concrete Semantics

```
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
```



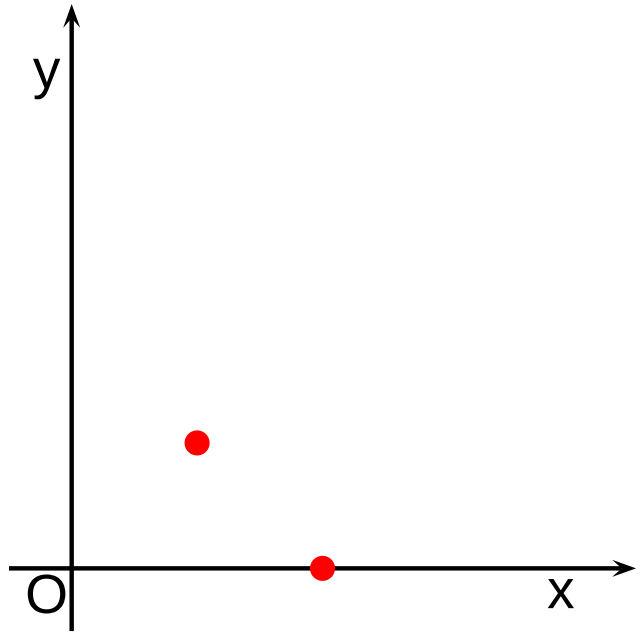
Example: The Concrete Semantics

```
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
```



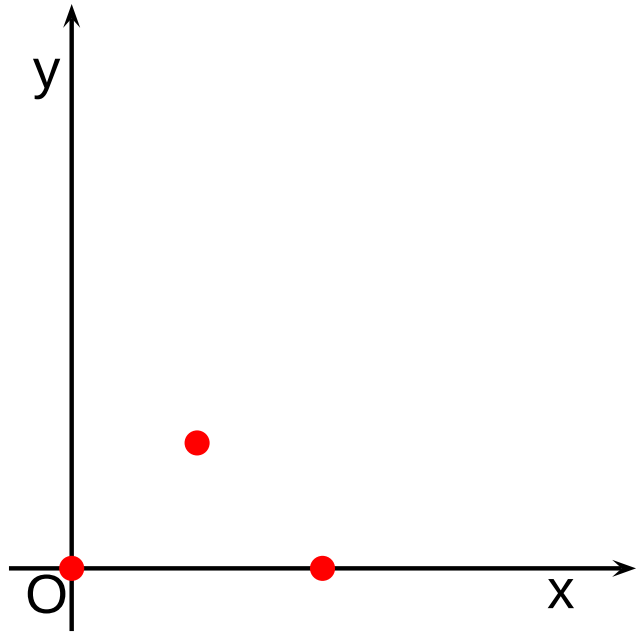
Example: The Concrete Semantics

```
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
```



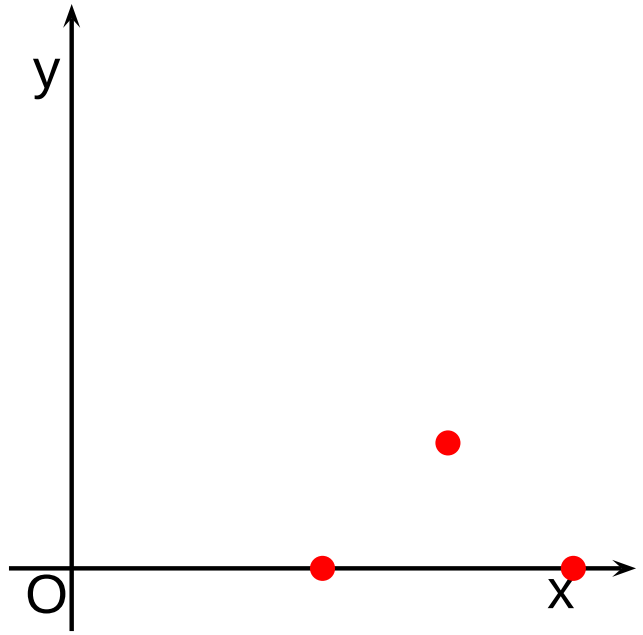
Example: The Concrete Semantics

```
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
```



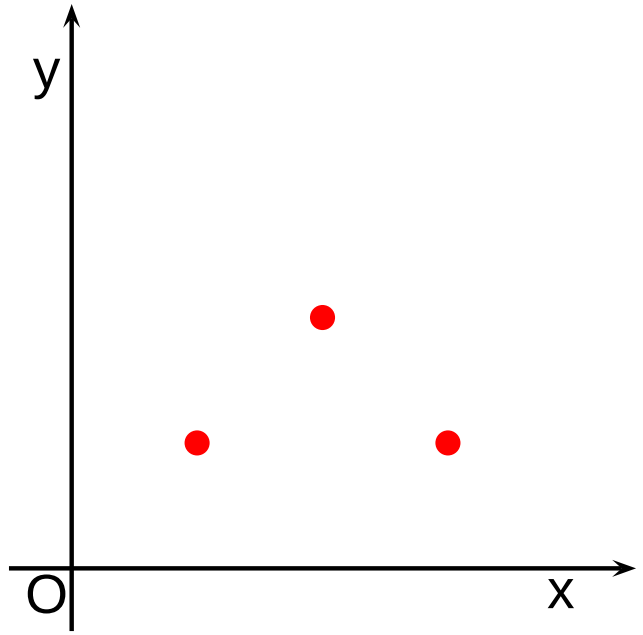
Example: The Concrete Semantics

```
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
```



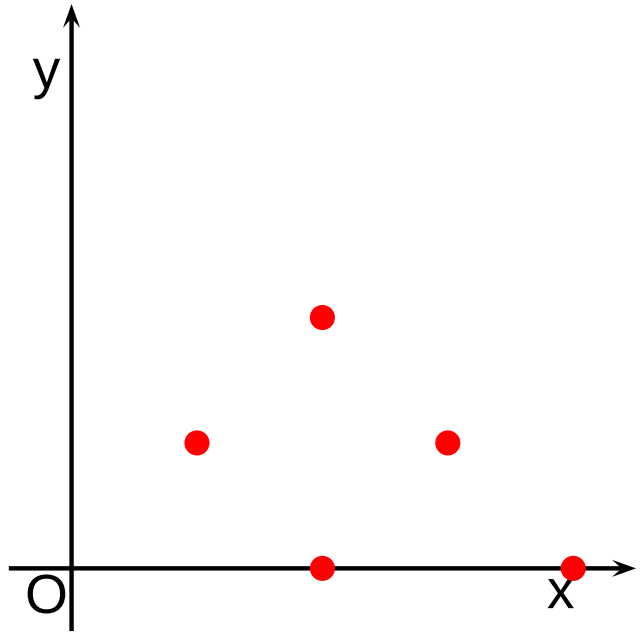
Example: The Concrete Semantics

```
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
```



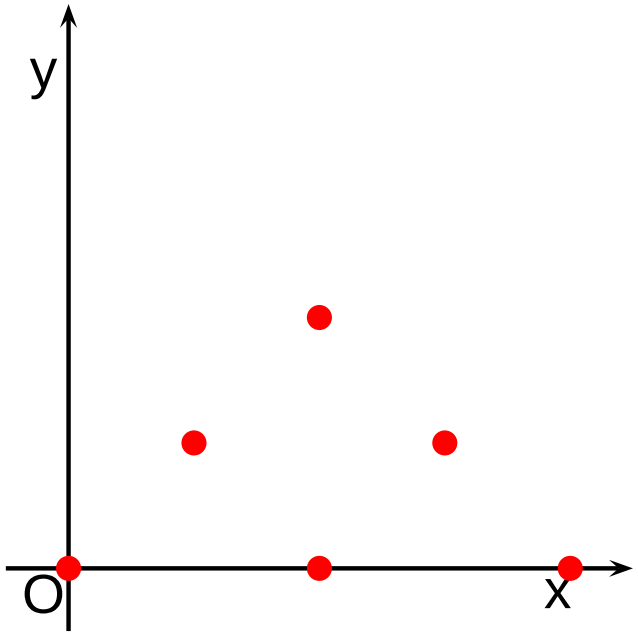
Example: The Concrete Semantics

```
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

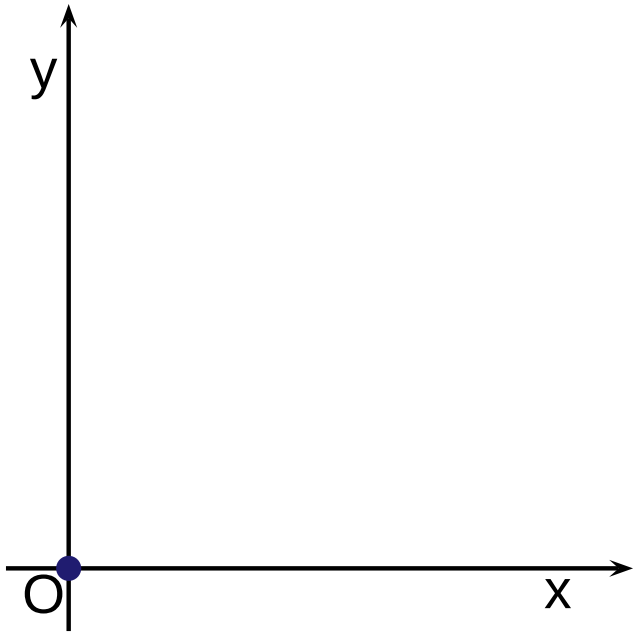
```
x := 0; y := 0;

while x <= 100 do
```

$\{1 = 0\}$

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

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

  endif

endwhile
```

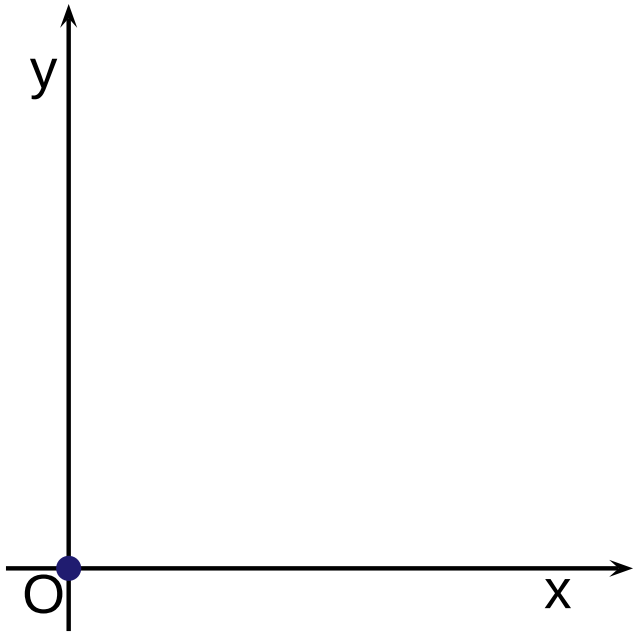
Example: The Abstract Semantics

```
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
```



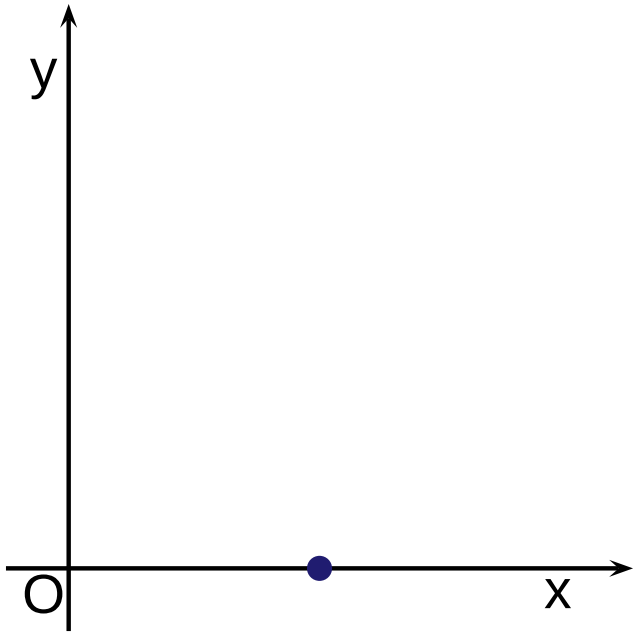
Example: The Abstract Semantics

```
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
```



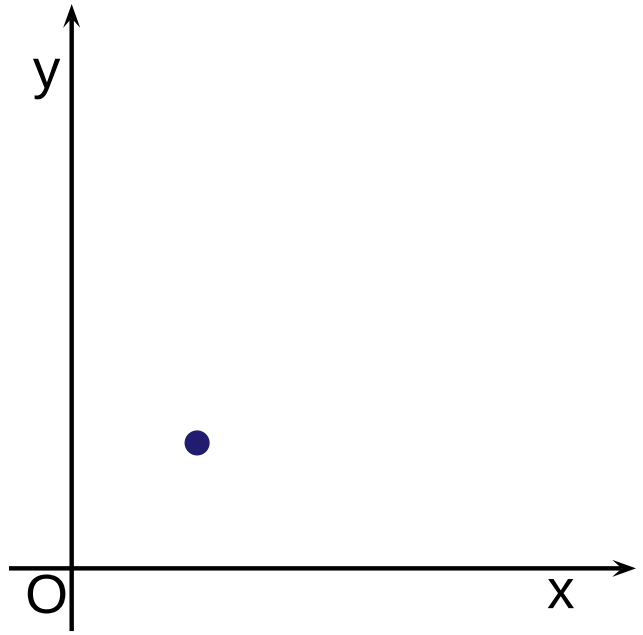
Example: The Abstract Semantics

```
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
```



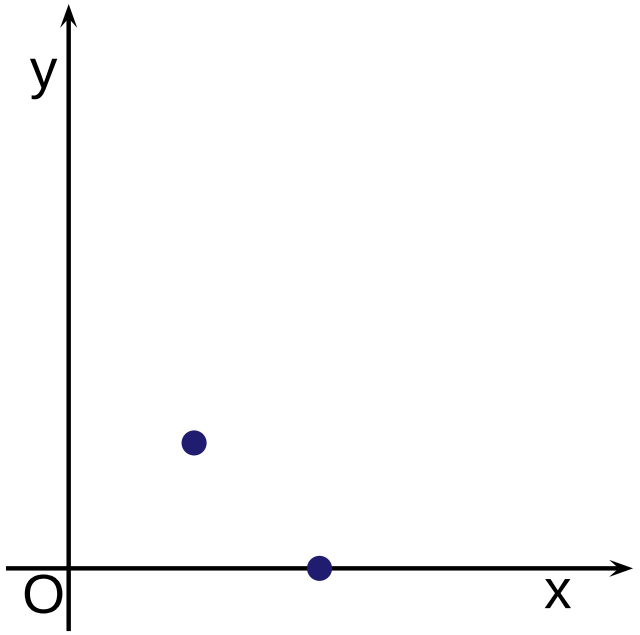
Example: The Abstract Semantics

```
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
```



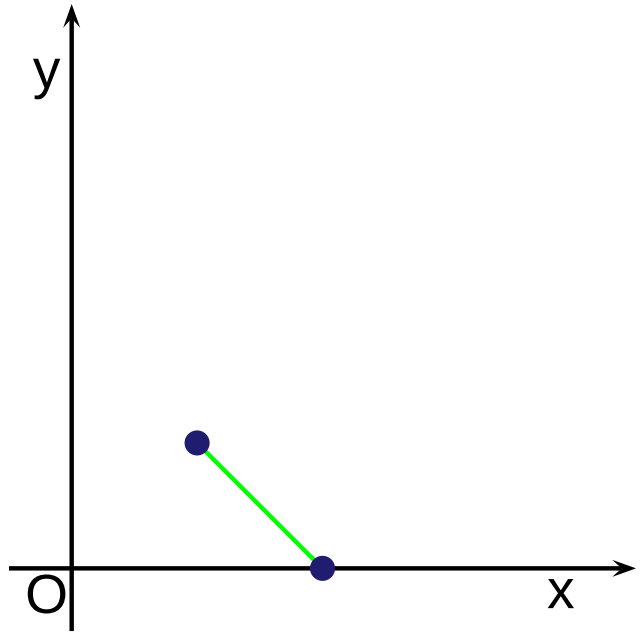
Example: The Abstract Semantics

```
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
```



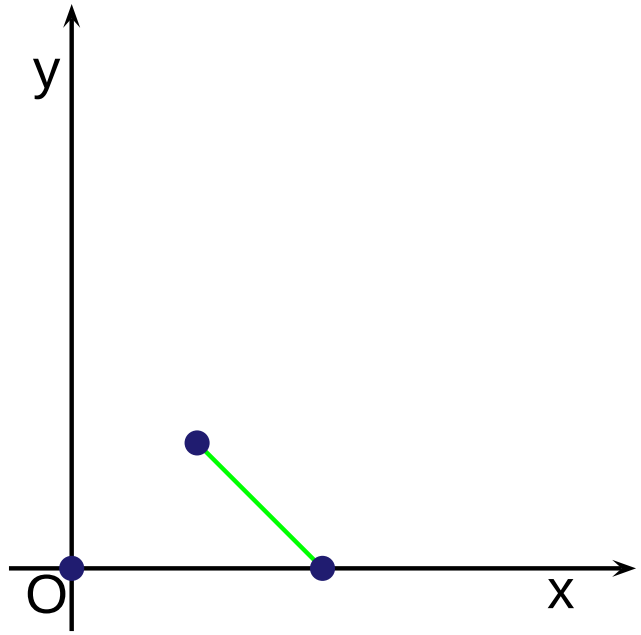
Example: The Abstract Semantics

```
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
```



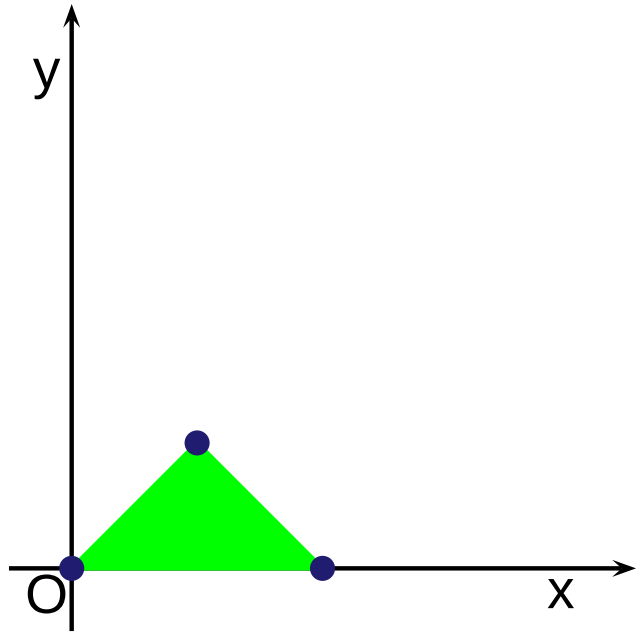
Example: The Abstract Semantics

```
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
```



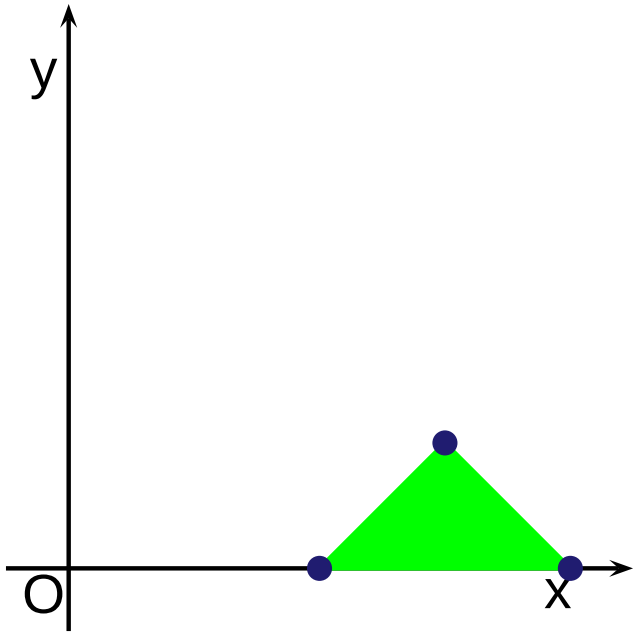
Example: The Abstract Semantics

```
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
```



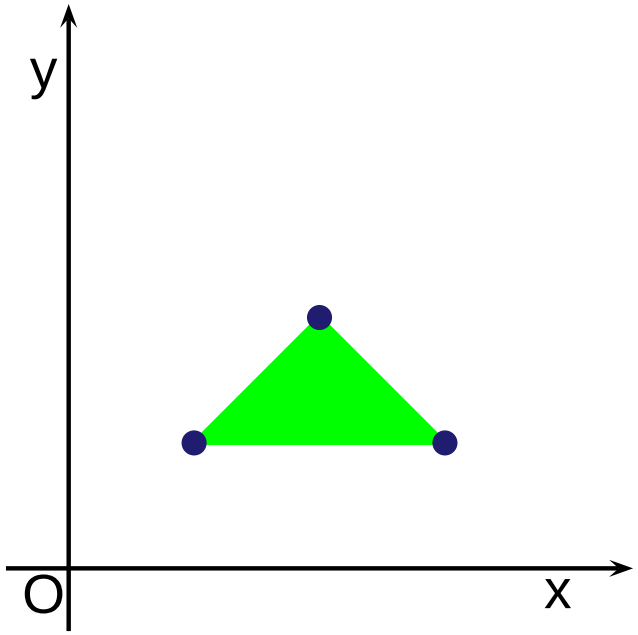
Example: The Abstract Semantics

```
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
```

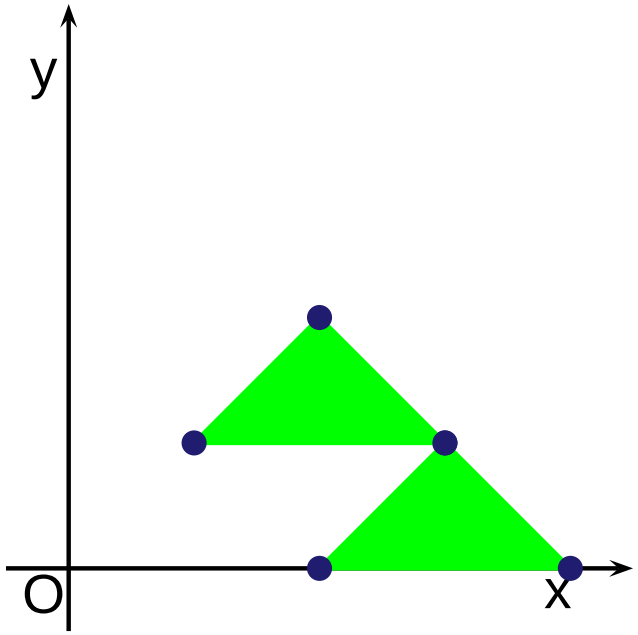


Example: The Abstract Semantics

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

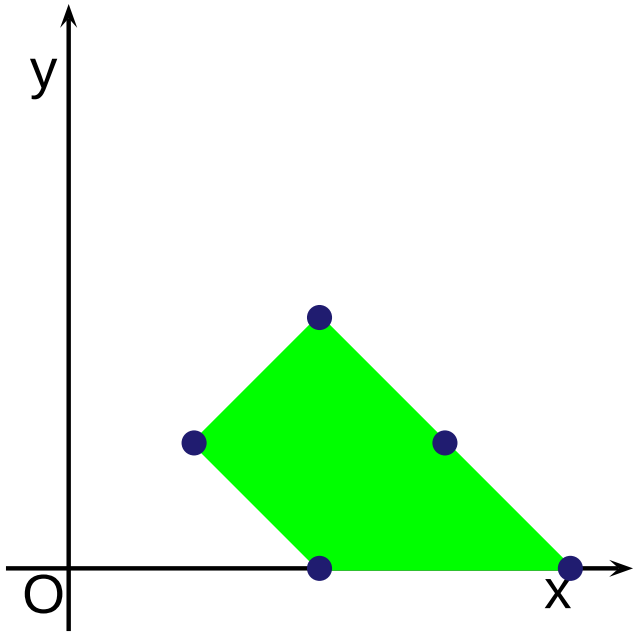
$\uplus \{1 \leq x \leq 2, x + y = 2\}$

```
endwhile
```



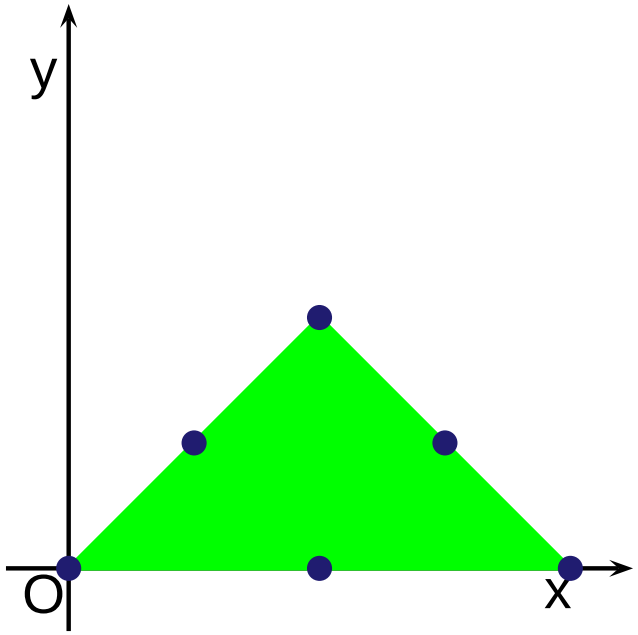
Example: The Abstract Semantics

```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x, x + y \leq 2\}$
```
  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: . . . AND SO ON . . . ?

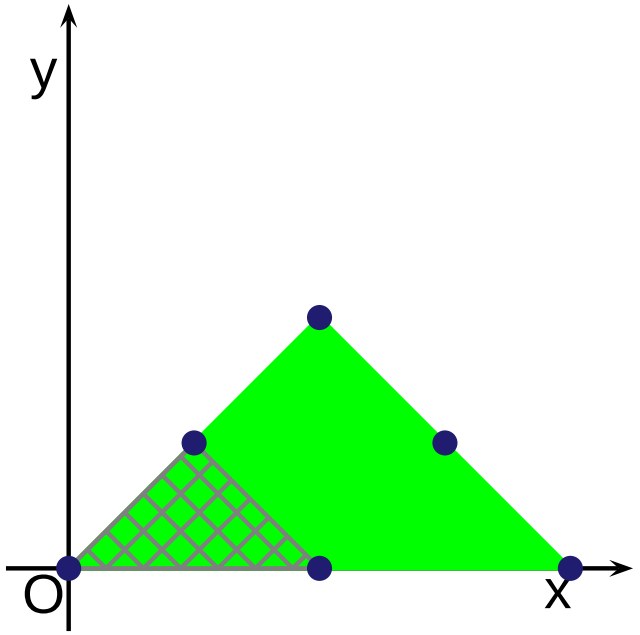
```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x, x + y \leq 4\}$
```
  read(b);
  if b then x := x+2
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
```
  else x := x+1; y := y+1;
```
$\{1 \leq y \leq x, x + y \leq 4\}$
```
  endif
```
$\{0 \leq y \leq x, 2 \leq x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



Example: Finite Convergence using Widening

```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x, x + y \leq 2\}$

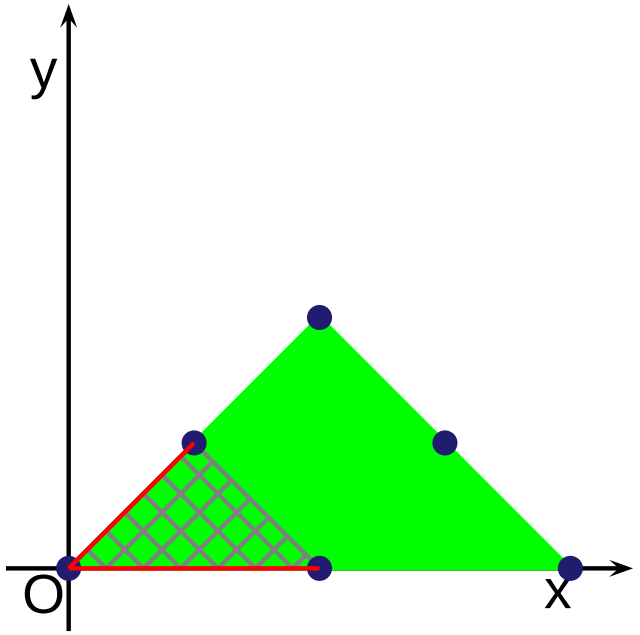
$\nabla \{0 \leq y \leq x, x + y \leq 4\}$
```
  read(b);
  if b then x := x+2
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
```
  else x := x+1; y := y+1;
```
$\{1 \leq y \leq x, x + y \leq 4\}$
```
  endif
```
$\{0 \leq y \leq x, 2 \leq x + y \leq 4\}$
```
endwhile
```



Example: Finite Convergence using Widening

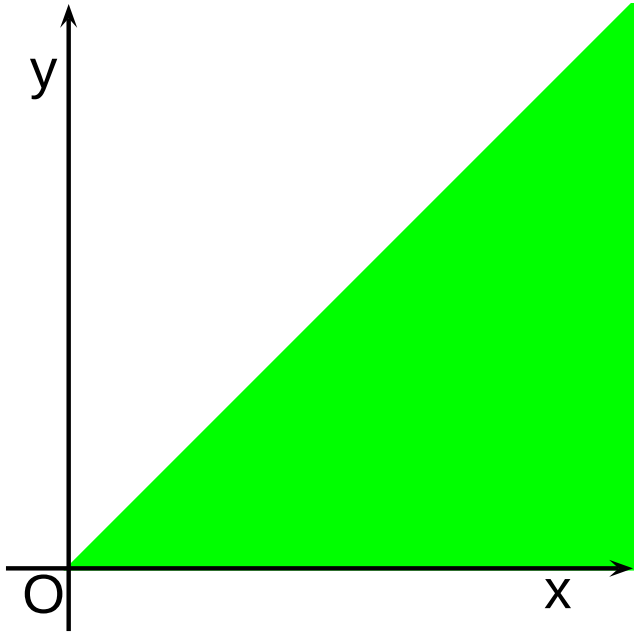
```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x, x + y \leq 2\}$

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



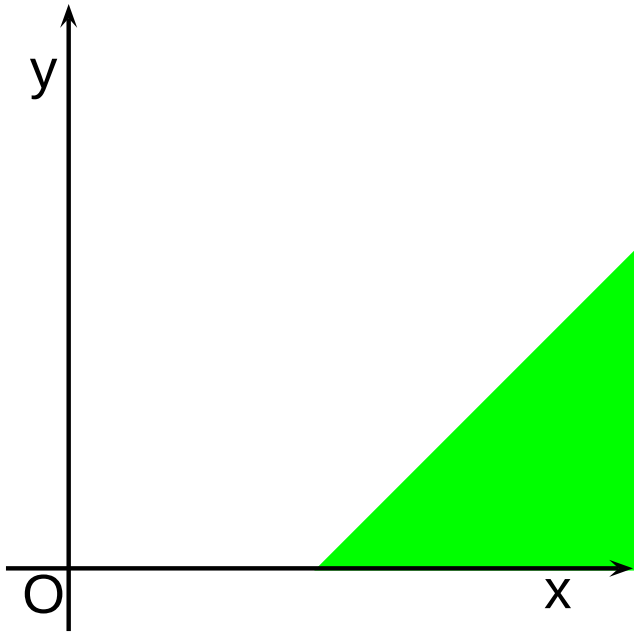
Example: An Abstract Post-Fixpoint

```
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
```



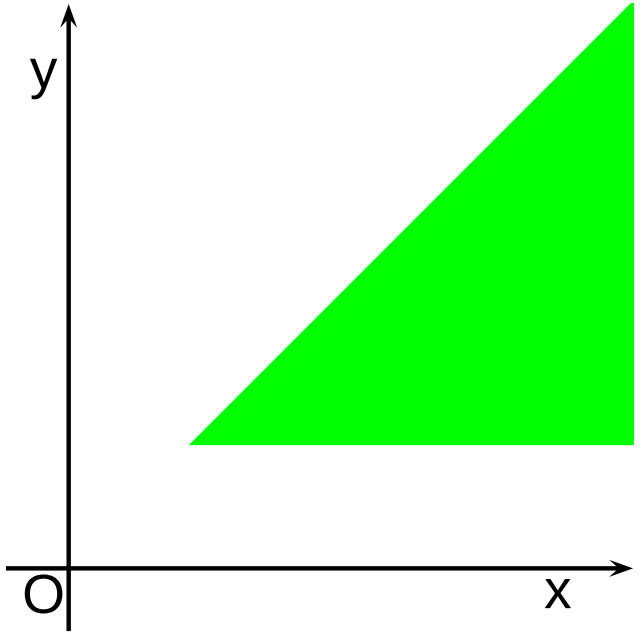
Example: Abstract Downward Iteration

```
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
```



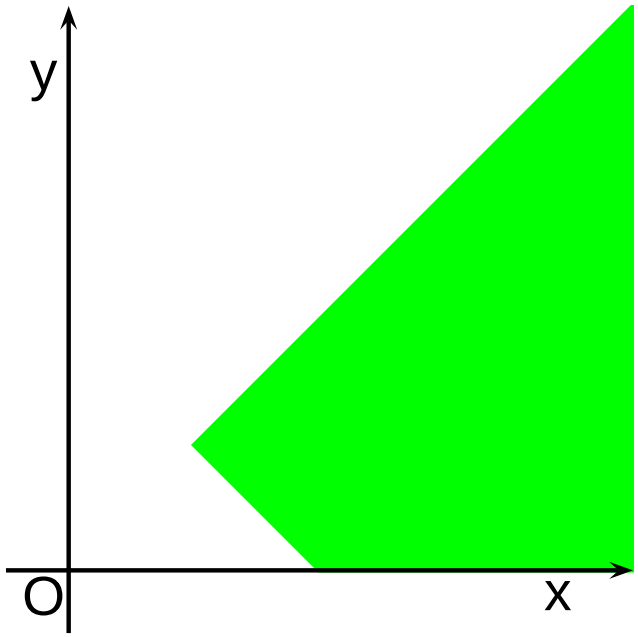
Example: Abstract Downward Iteration

```
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
```



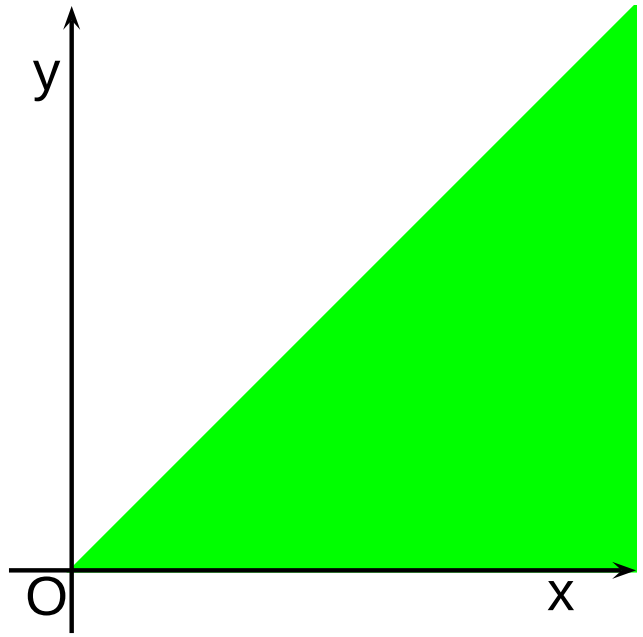
Example: Abstract Downward Iteration

```
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

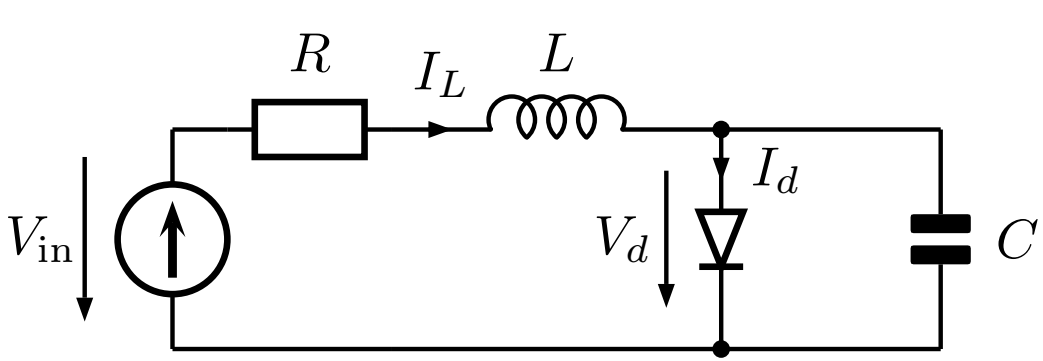
```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x \leq 100\}$
```
  read(b);
  if b then x := x+2
```
$\{0 \leq y \leq x - 2\}$
```
  else x := x+1; y := y+1;
```
$\{1 \leq y \leq x\}$
```
  endif
```
$\{0 \leq y \leq x, 2 \leq x + y\}$
```
endwhile
```

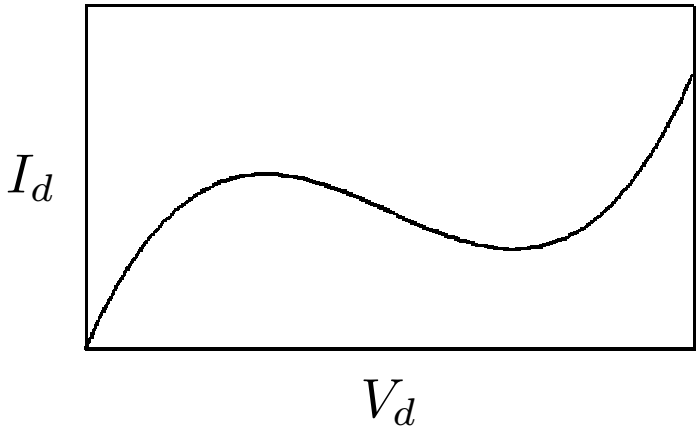


Example: Abstract Fixpoint

```
x := 0; y := 0;
```
$\{x = 0, y = 0\}$
```
while x <= 100 do
```
$\{0 \leq y \leq x \leq 100\}$
```
  read(b);
  if b then x := x+2
```
$\{0 \leq y \leq x - 2 \leq 100\}$
```
  else x := x+1; y := y+1;
```
$\{1 \leq y \leq x \leq 101\}$
```
  endif
```
$\{0 \leq y \leq x \leq 102, 2 \leq x + y \leq 202\}$
```
endwhile
```
$\{100 < x \leq 102, 0 \leq y \leq x, x + y \leq 202\}$

Verification of Analog Circuits: Cyclic Invariants





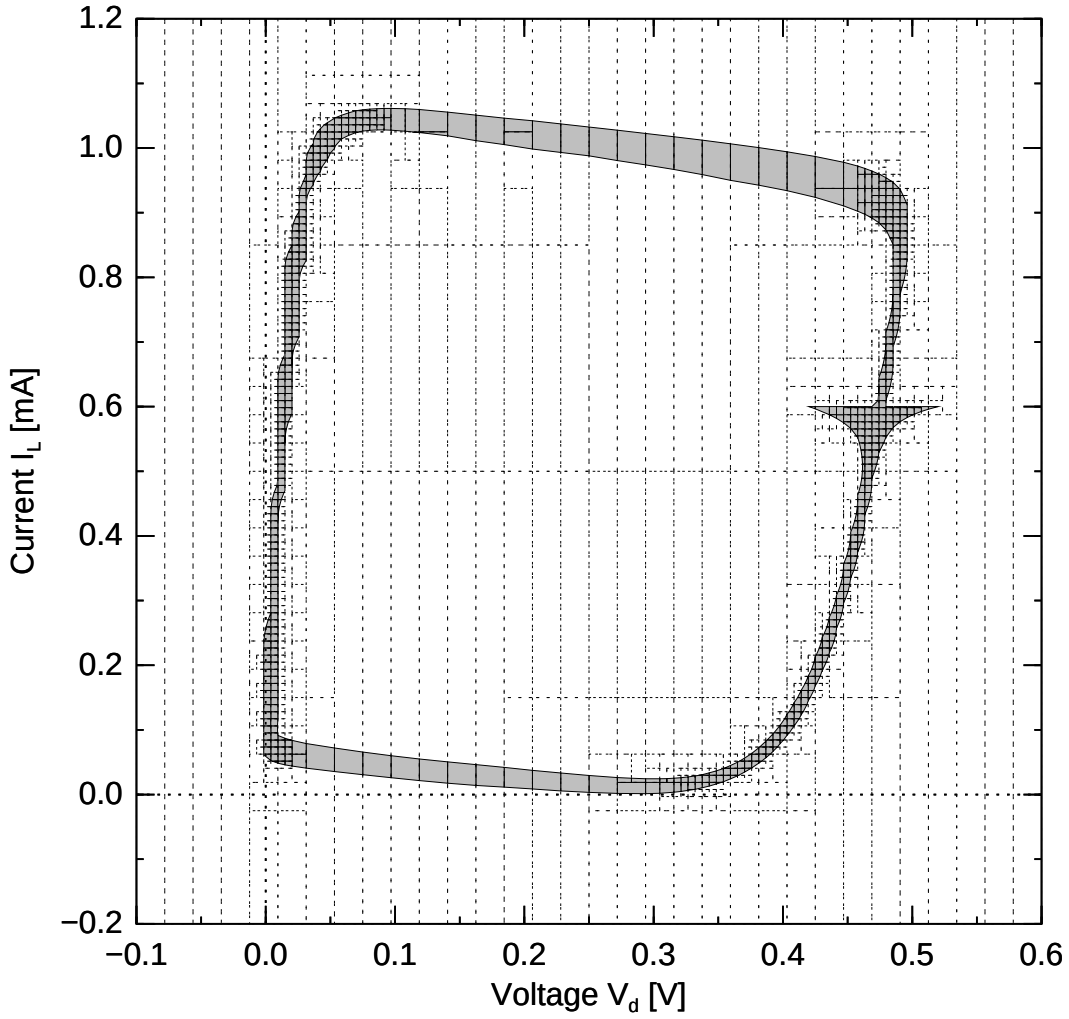
The system is described by the second-order state equations

$$\dot{V}_d = 1/C\big(-I_d(V_d) + I_L\big),$$
$$\dot{I}_L = 1/L(-V_d - RI_L + V_{\text{in}}).$$

Verification of Analog Circuits (cont'd)

- Frehse shows how a cyclic invariant can be obtained for this circuit using the PHAVer system.
- First, a piecewise affine envelope is constructed for the tunnel diode characteristic $I_d(V_d)$: sufficient precision is obtained by subdividing the range $V_d \in [-0.1\,\mathrm{V}, 0.6\,\mathrm{V}]$ into 64 intervals.
- Forward reachability computation allows to prove that the set of initial states corresponding to $V_d \in [0.42\,\mathrm{V}, 0.52\,\mathrm{V}]$ and $I_L = 0.6\,\mathrm{mA}$ gives rise to a cycle.

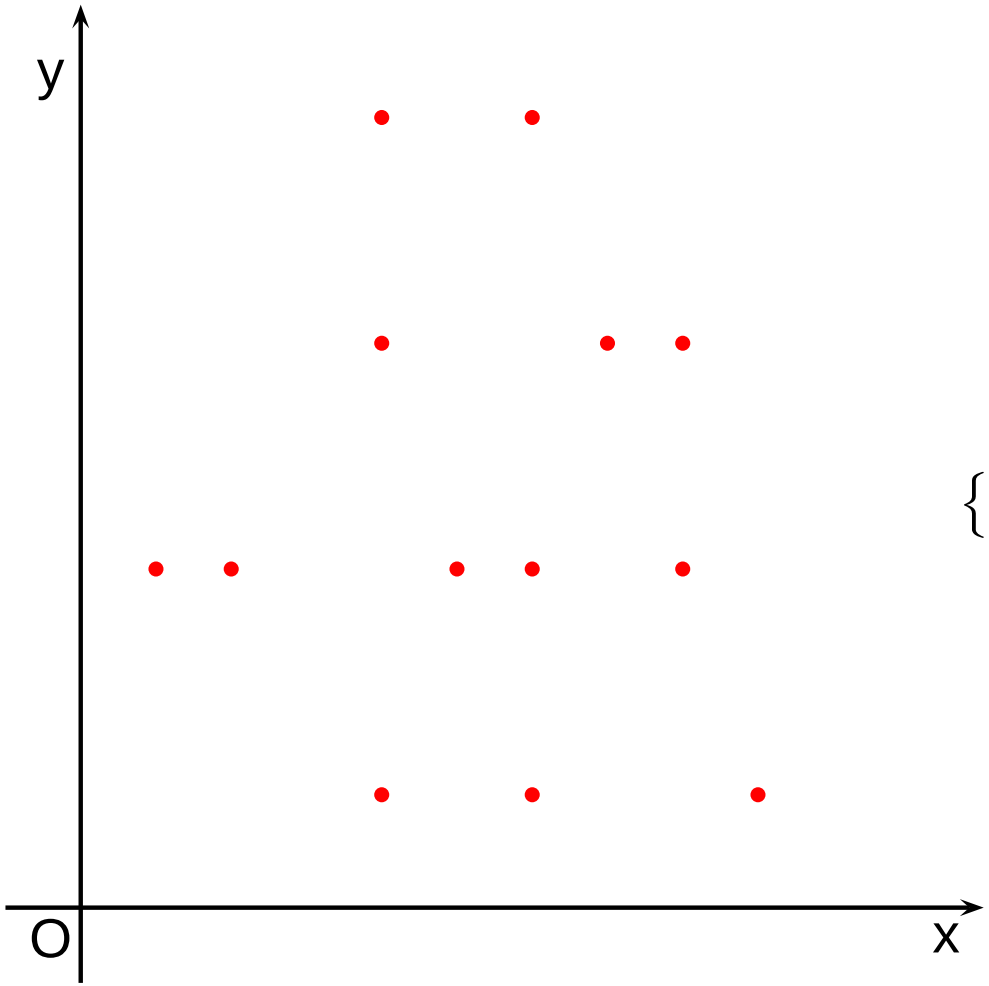
Verification of Analog Circuits (cont'd)



Numerical Domains and Software/Hardware Verification

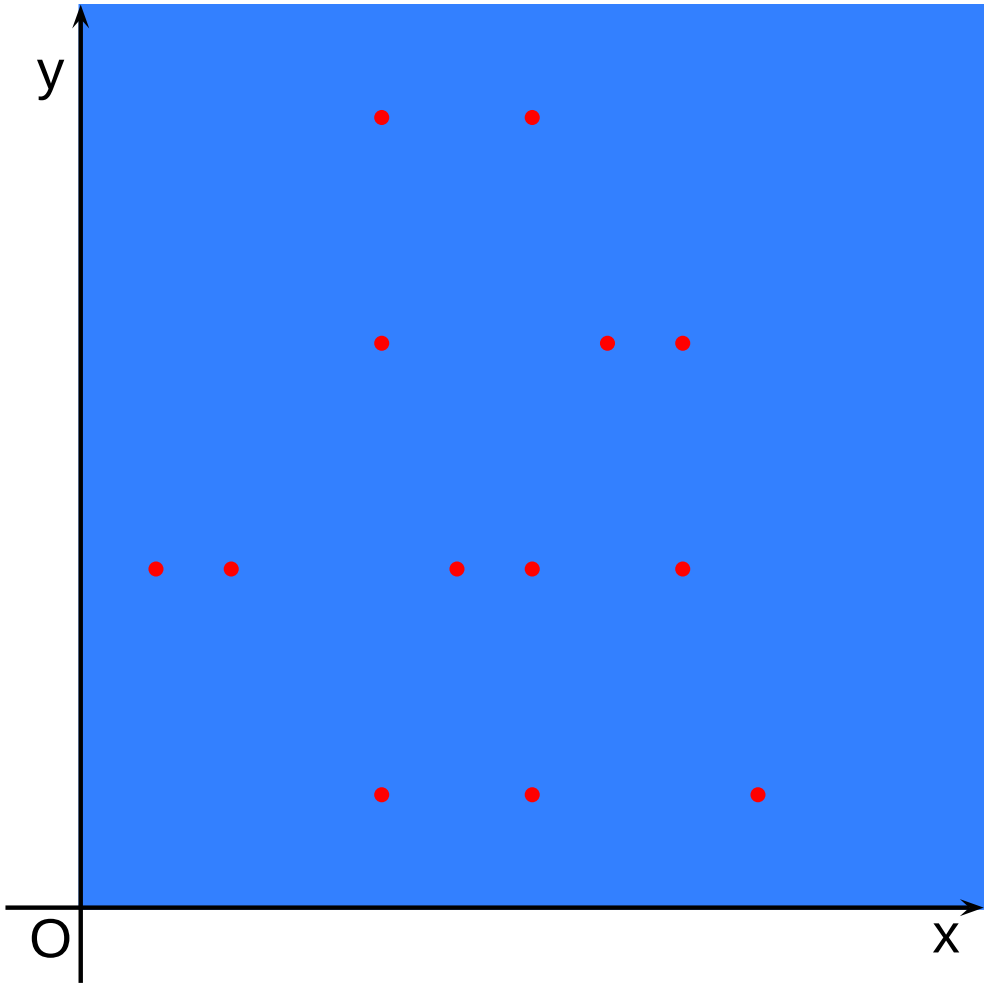
- The complexity/precision tradeoff is particularly acute in the verification of software/hardware.
- Different domains (e.g., different classes of polyhedra) are required to face different situations.
- Conservative approximation preserves soundness, but may prevent the derivation of the properties of interest.

Numerical Abstractions: No Abstraction



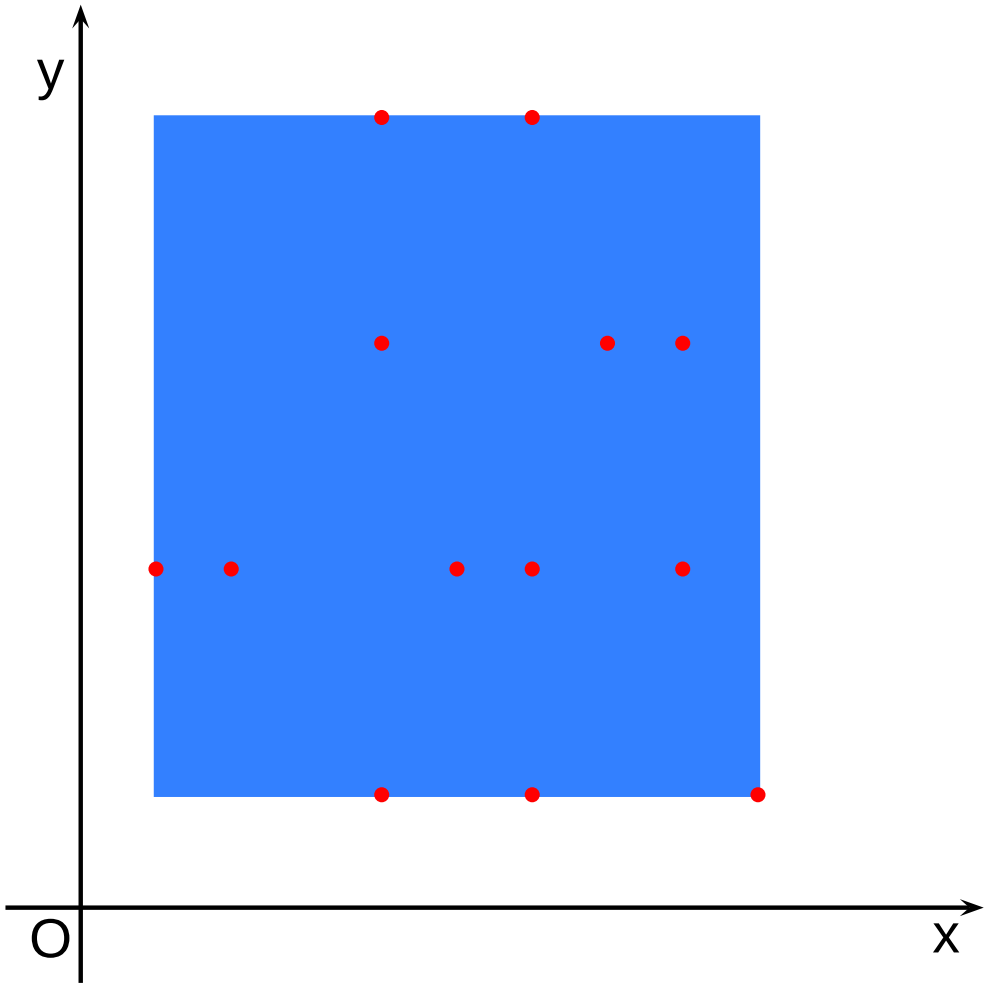
$$\{\ldots,(2,9),\ldots,(12,21)\ldots,\}$$

Numerical Abstractions: Signs



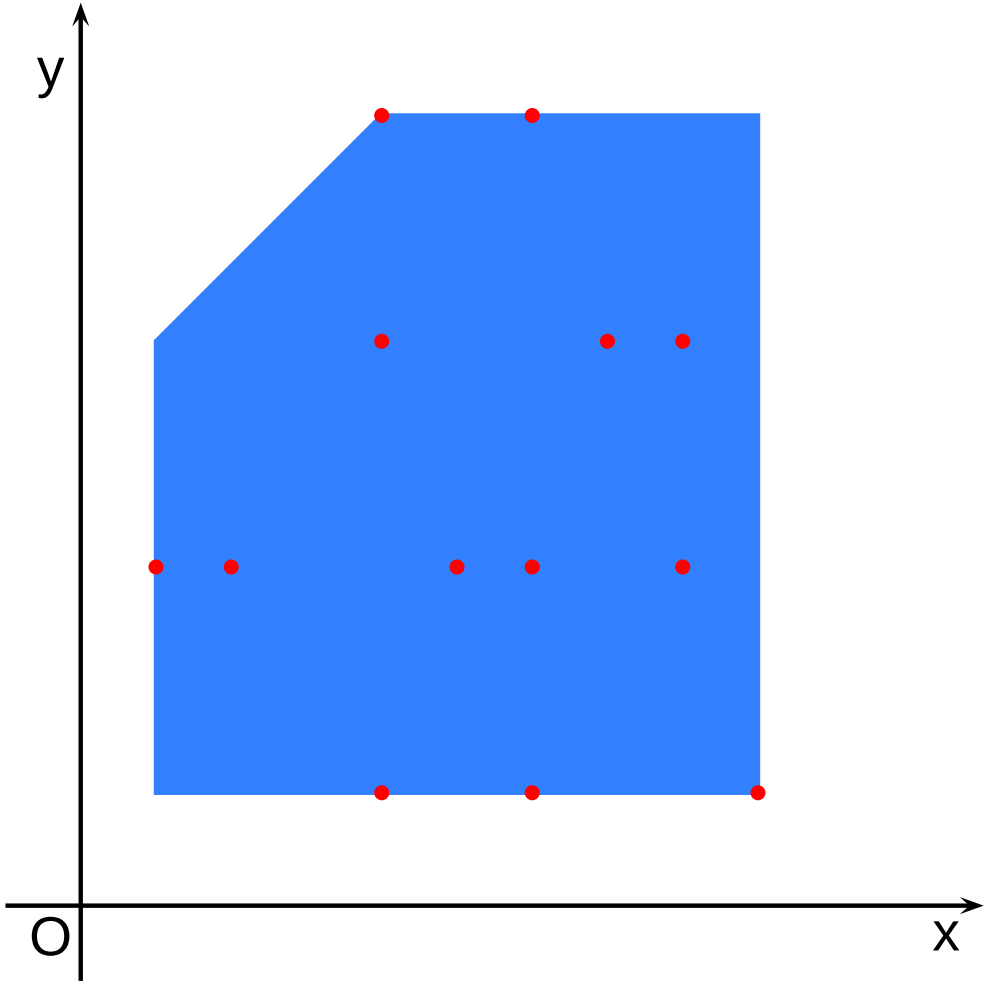
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Numerical Abstractions: Bounding Boxes



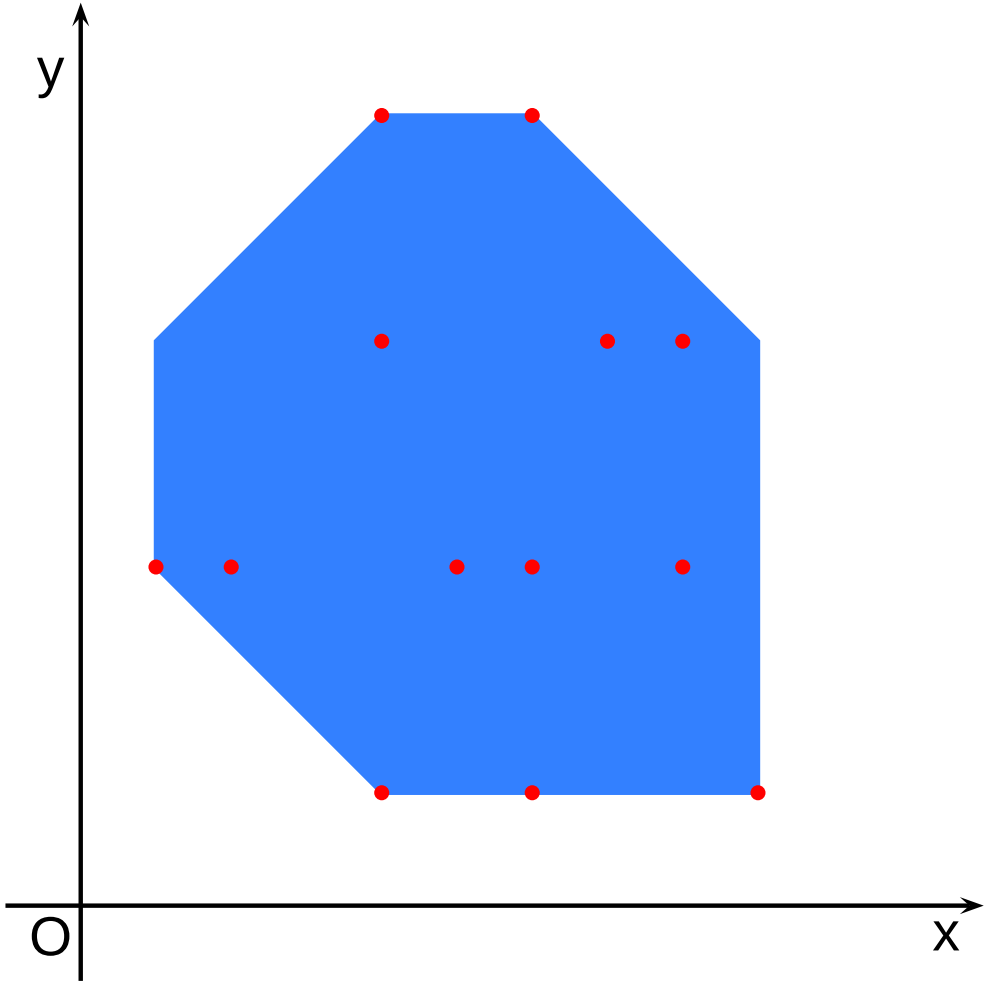
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \end{cases}$$

Numerical Abstractions: Bounded Differences



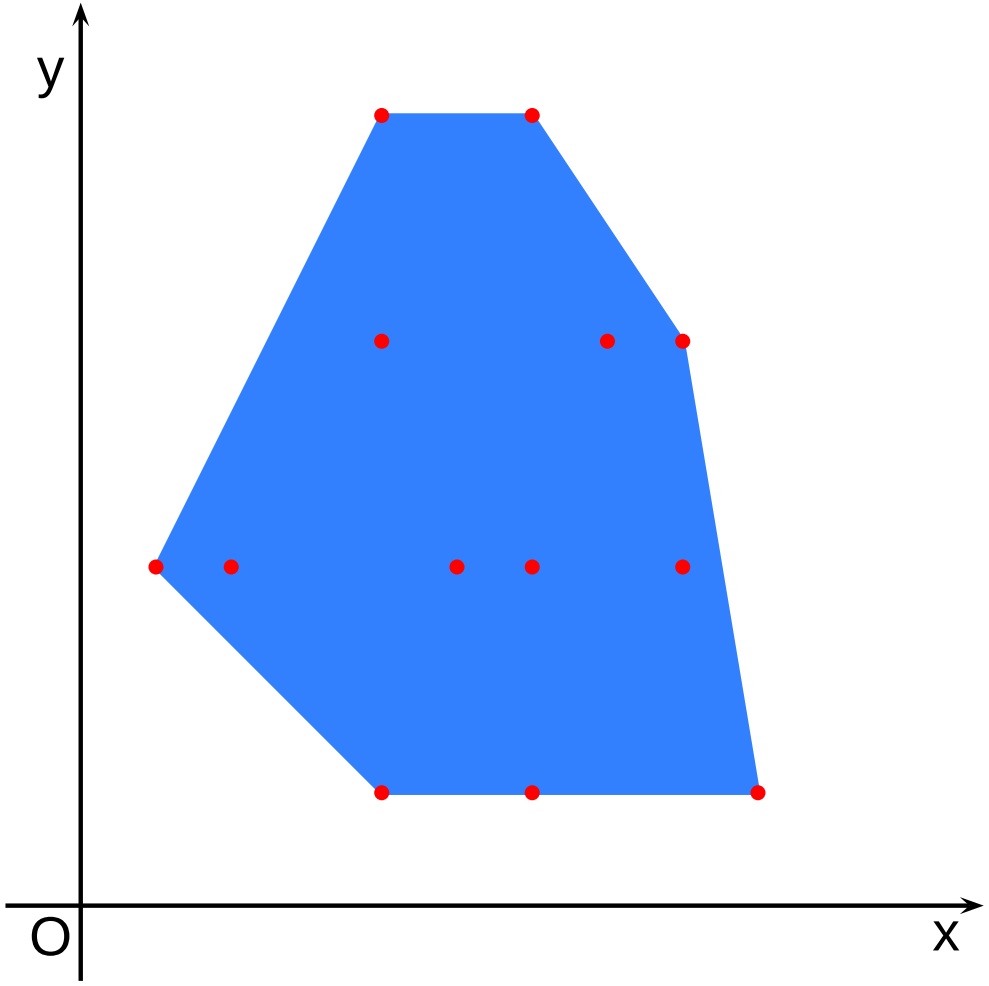
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \end{cases}$$

Numerical Abstractions: Octagons



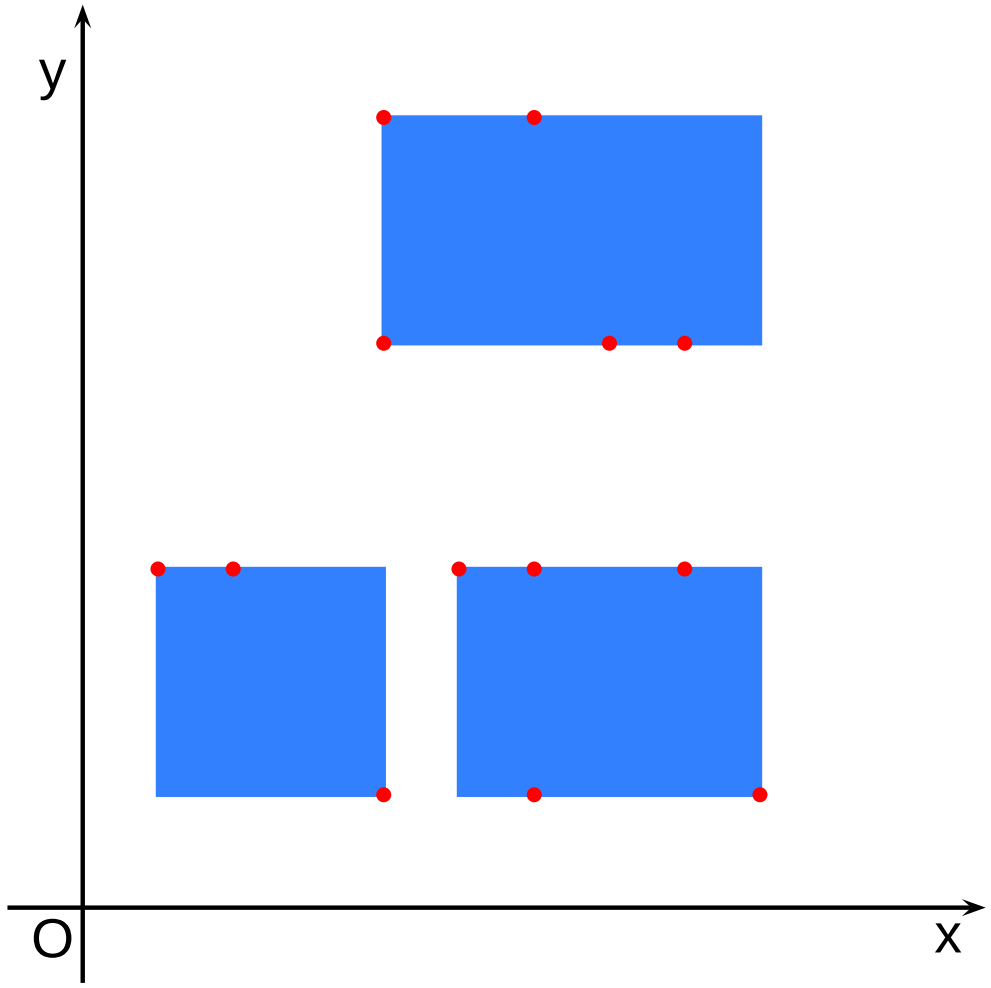
$$\begin{cases} 2 \leq x \leq 18 \\ 3 \leq y \leq 21 \\ -10 \leq x - y \\ 11 \leq x + y \leq 33 \end{cases}$$

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: Powersets of Bounding Boxes

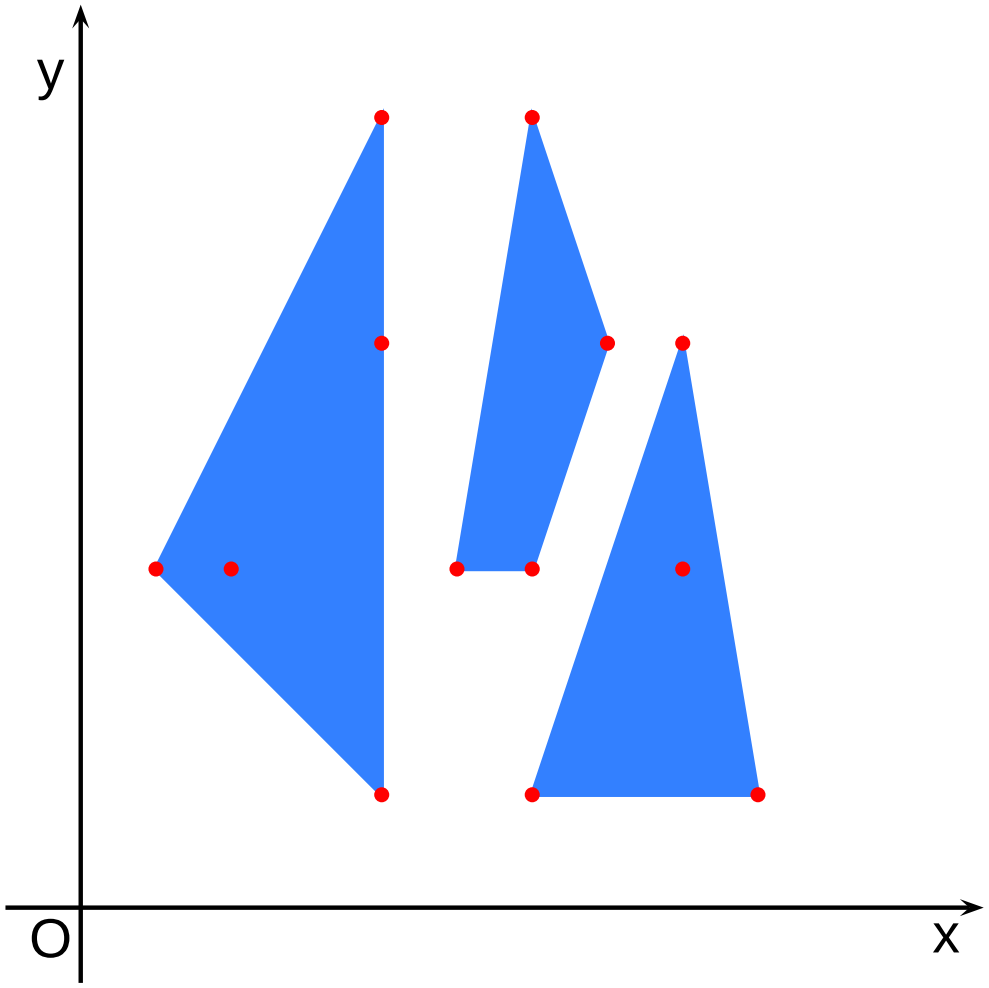


$$\begin{cases} 2 \leq x \leq 8 \\ 3 \leq y \leq 9 \end{cases}$$

$$\begin{cases} 10 \leq x \leq 18 \\ 3 \leq y \leq 9 \end{cases}$$

$$\begin{cases} 8 \leq x \leq 18 \\ 15 \leq y \leq 21 \end{cases}$$

Numerical Abstractions: Powersets of Polyhedra

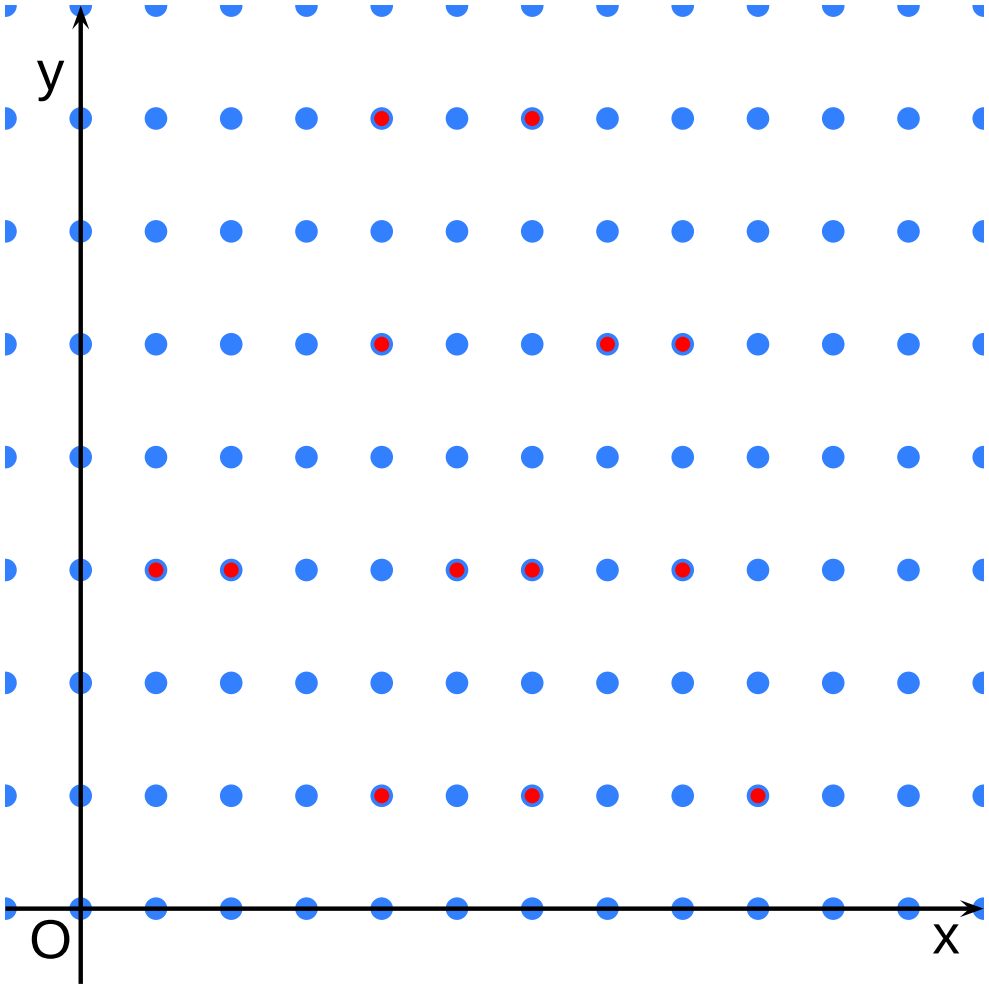


$$\begin{cases} 6x - y \geq 51 \\ 3x - y \geq 27 \\ 3x + y \leq 57 \\ y \geq 9 \end{cases}$$

$$\begin{cases} x + y \geq 11 \\ 2x - y \geq -5 \\ x \leq 8 \end{cases}$$

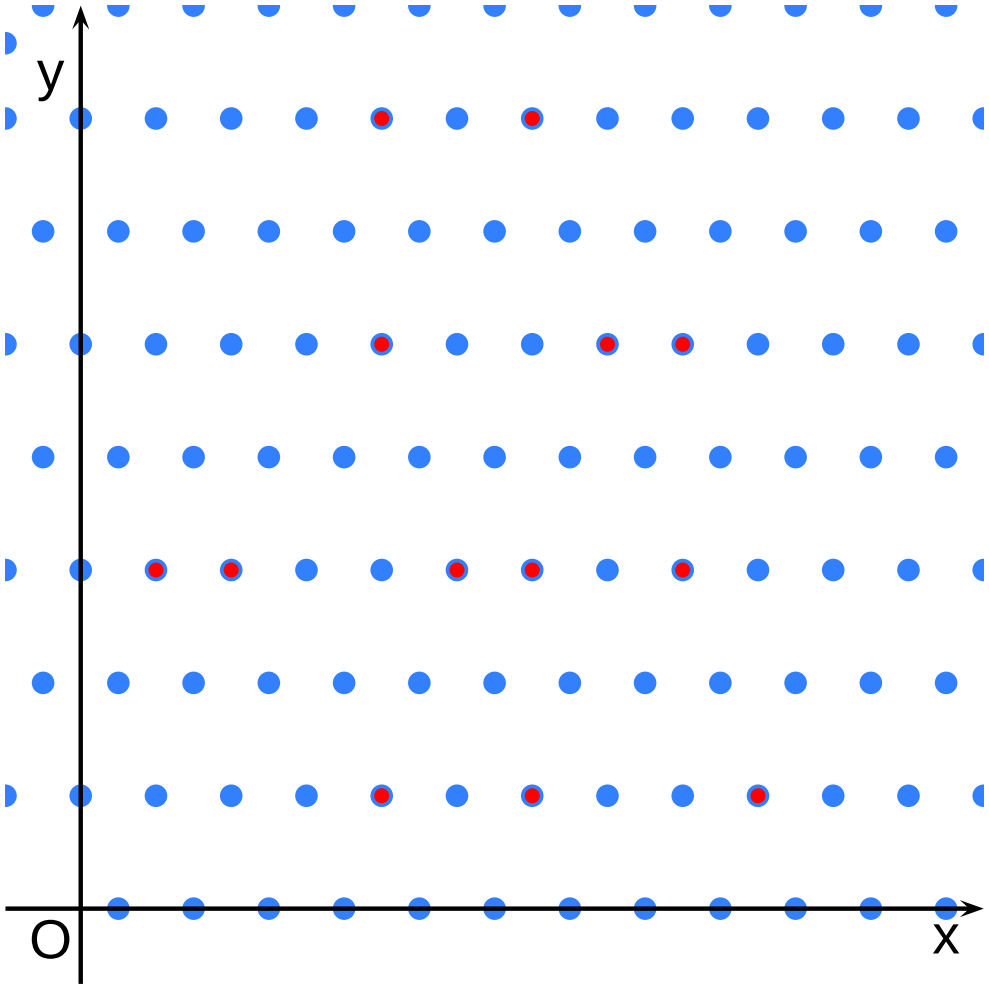
$$\begin{cases} 6x + y \leq 111 \\ 3x - y \geq 33 \\ y \geq 3 \end{cases}$$

Numerical Abstractions: Non-Relational Grids



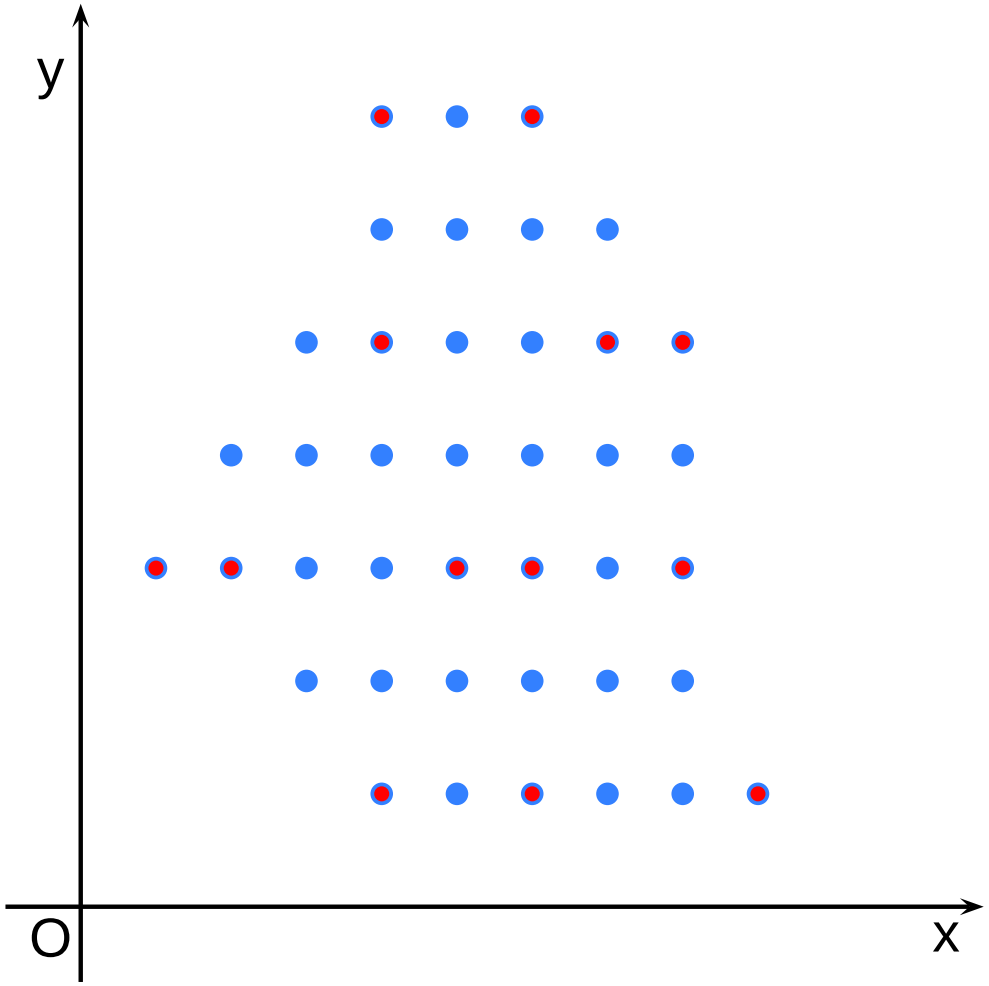
$$\begin{cases} x = 0 \mod 2 \\ y = 0 \mod 3 \end{cases}$$

Numerical Abstractions: Relational Grids



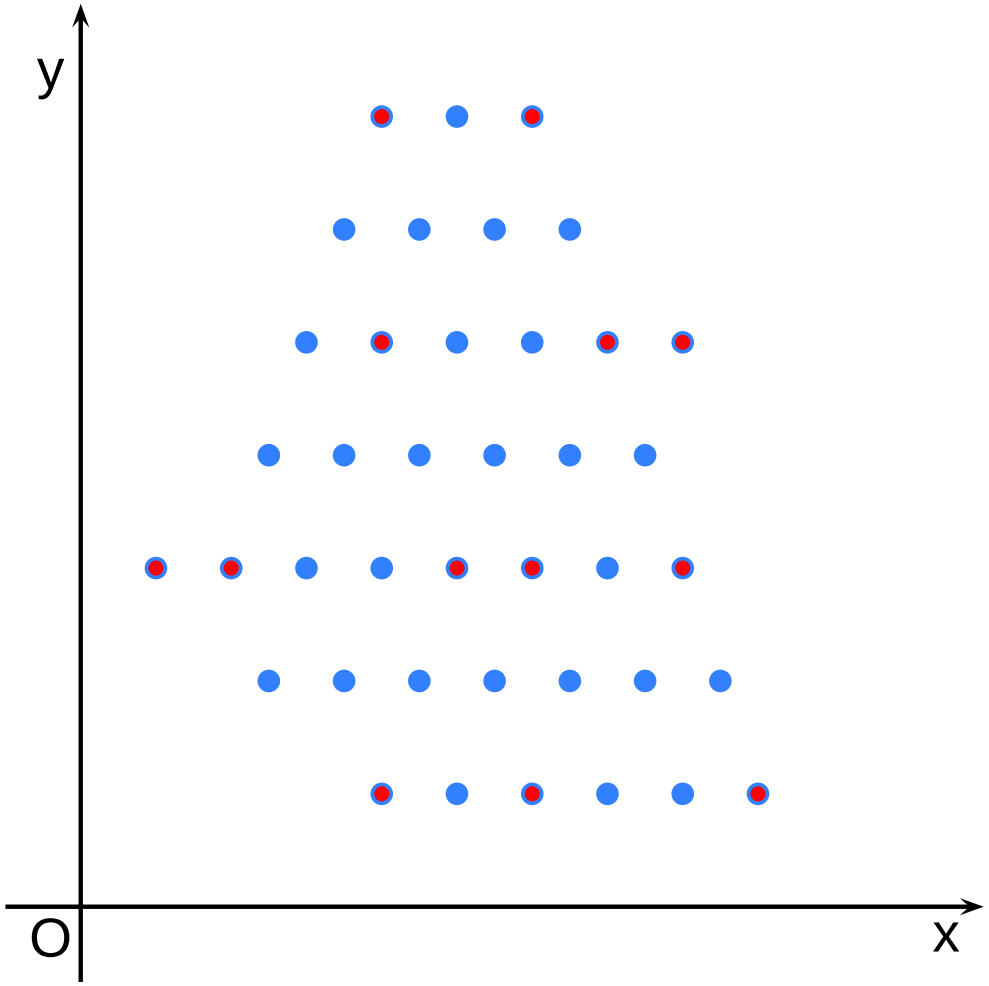
$$\begin{cases} x + y = 1 \mod 2 \\ y = 0 \mod 3 \end{cases}$$

Numerical Abstractions: Grid-Polyhedra (I)



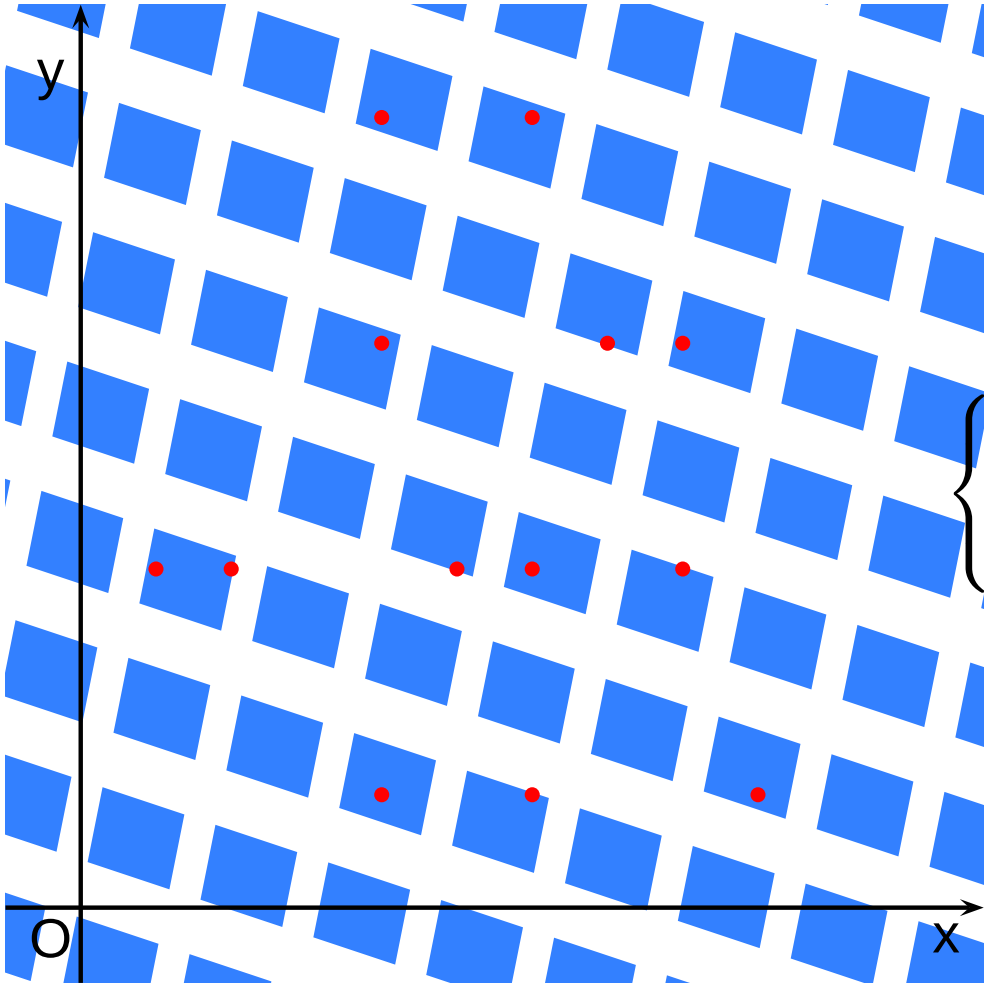
$$\begin{cases} x = 0 \mod 2 \\ y = 0 \mod 3 \\ 6x + y \leq 111 \\ 3x + 2y \leq 78 \\ x + y \geq 11 \\ 2x - y \geq -5 \\ y \geq 3 \\ y \leq 21 \end{cases}$$

Numerical Abstractions: Grid-Polyhedra (II)



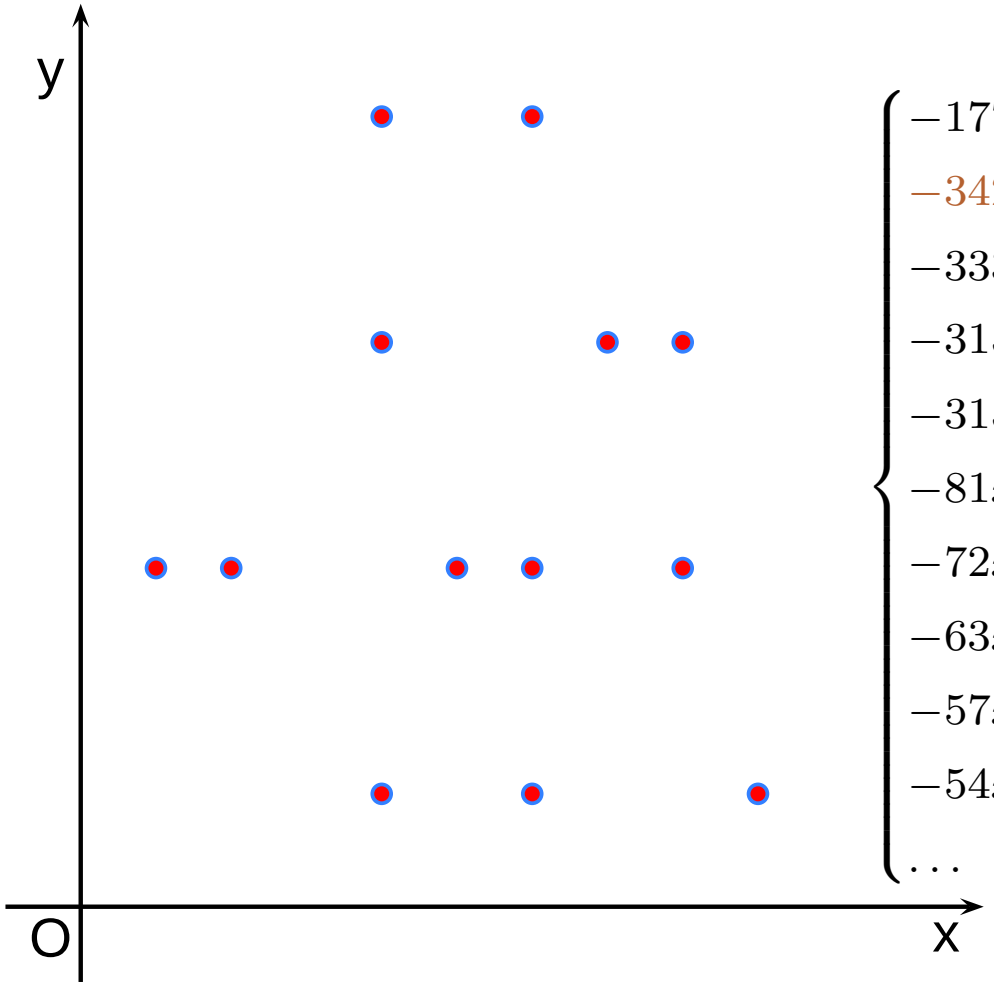
$$\begin{cases} x + y = 1 \mod 2 \\ y = 0 \mod 3 \\ 6x + y \leq 111 \\ 3x + 2y \leq 78 \\ x + y \geq 11 \\ 2x - y \geq -5 \\ y \geq 3 \\ y \leq 21 \end{cases}$$

Numerical Abstractions: Trapezoidal Congruences



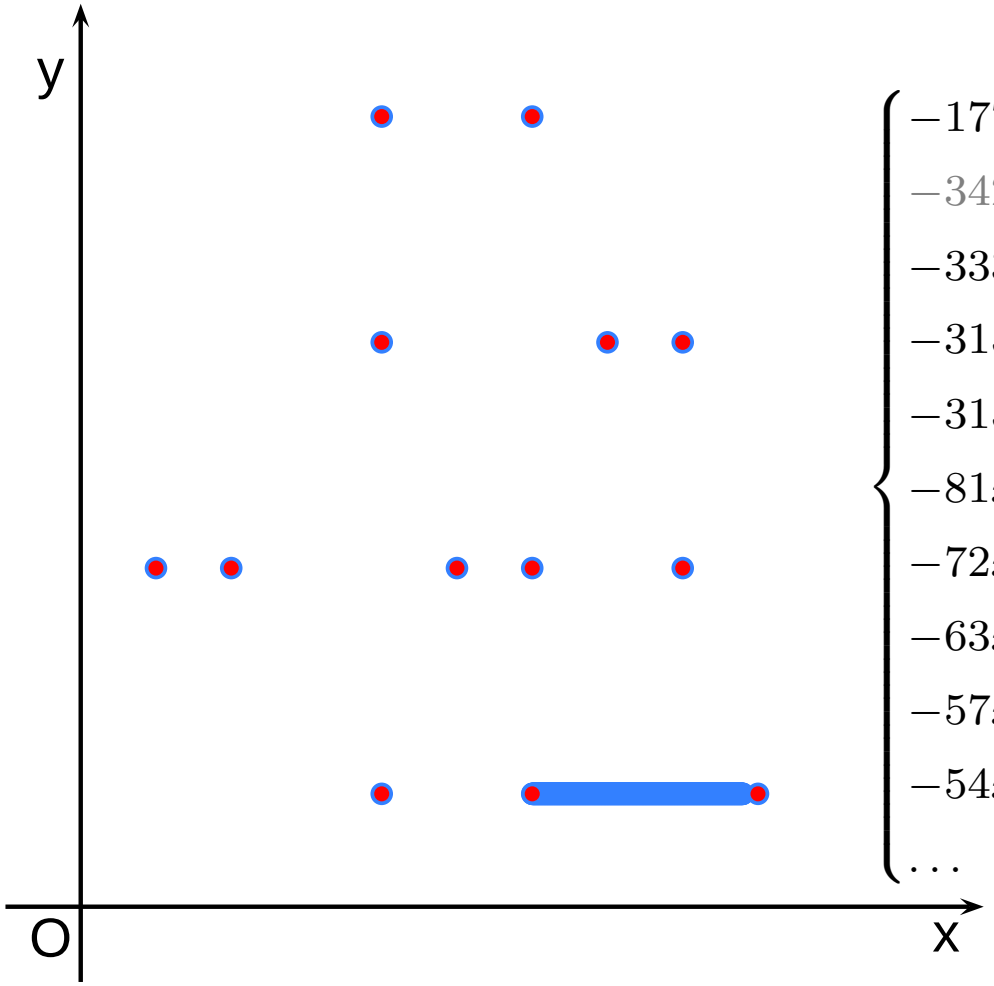
$$\begin{cases} x + 3y \in [4, 10] & \mod 11 \\ 5x - y \in [0, 11] & \mod 16 \end{cases}$$

Numerical Abstractions: Polynomial Cones (I)



$$\begin{cases} -1773x - 1176y + 45x^2 + 57xy + 20y^2 \geq -17 \\ -342x - 348y + 9x^2 + 24xy + 7y^2 \geq -2925 \\ -333x - 216y + 9x^2 + 9xy + 4y^2 \geq -3348 \\ -315x + 156y + 18x^2 - 15xy + 2y^2 \geq -1242 \\ -315x + 228y + 18x^2 - 15xy - 2y^2 \geq -1062 \\ -81x - 32y + 3x^2 + 7xy - y^2 \geq -393 \\ -72x - 32y + 3x^2 + 4xy \geq -384 \\ -63x - 8y + 3x^2 + xy \geq -312 \\ -57x + 12y + 3x^2 - xy \geq -252 \\ -54x - 168y + 6xy + 5y^2 \geq -1107 \\ \dots \end{cases}$$

Numerical Abstractions: Polynomial Cones (II)



$$
\begin{cases}
-1773x - 1176y + 45x^2 + 57xy + 20y^2 \geq -17 \\
-342x - 348y + 9x^2 + 24xy + 7y^2 \geq -2925 \\
-333x - 216y + 9x^2 + 9xy + 4y^2 \geq -3348 \\
-315x + 156y + 18x^2 - 15xy + 2y^2 \geq -1242 \\
-315x + 228y + 18x^2 - 15xy - 2y^2 \geq -1062 \\
-81x - 32y + 3x^2 + 7xy - y^2 \geq -393 \\
-72x - 32y + 3x^2 + 4xy \geq -384 \\
-63x - 8y + 3x^2 + xy \geq -312 \\
-57x + 12y + 3x^2 - xy \geq -252 \\
-54x - 168y + 6xy + 5y^2 \geq -1107 \\
\ldots
\end{cases}
$$

PARMA POLYHEDRA LIBRARY

- Starting from 2001, we have developed a library for polyhedral computations especially targeting analysis and verification tasks.
- This line of work resulted in several innovations: new domains, new algorithms, new widenings...
- It is now used, among others, by GCC (the GNU Compiler Collection): if you use a system that comes with GCC, you probably have the PPL already installed.
- Still under very active development:
  - support for the approximation of floating point computations;
  - support for the approximation of bounded arithmetic;
  - parametric integer programming (thanks to UVSQ);
  - ...
- For more information:

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

Part 3

Recent and Ongoing Work

Detecting Exact Joins

- When using a numerical abstract domain, the computation of joins is a major source of precision losses. Possible workarounds include:
  - delay the computation of joins (e.g., trace partitioning);
  - avoid the computation of joins (e.g., disjunctive domains).
- Sometimes no precision loss is allowed (e.g., loop parallelizations).
- Disjunctive sets of domain elements: the fewer elements, the better.
- For $\{D_1, \ldots, D_k\} \subseteq \mathbb{D}_n$, decide whether $\biguplus_{i=1}^{k} D_i = \bigcup_{i=1}^{k} D_i$.
- Too hard! But the binary case is doable: decide whether

$$D_1 \uplus D_2 = D_1 \cup D_2.$$

R. Bagnara, P. M. Hill, E. Zaffanella.
Exact Join Detection for Convex Polyhedra and Other Numerical Abstractions
To appear in *Computational Geometry: Theory and Applications*.

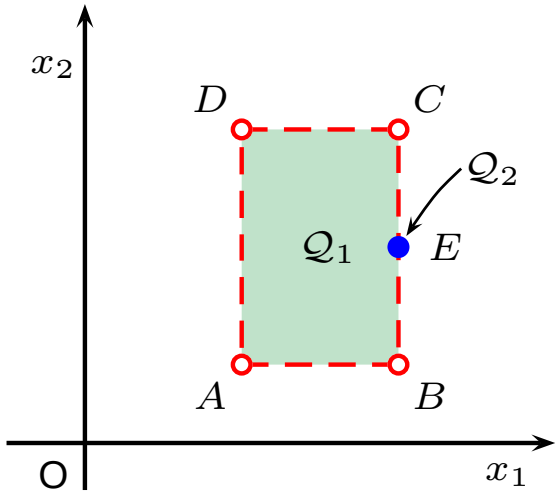
Exact Joins for Convex Polyhedra

- Problem already studied by (among others) Bemporad, Fukuda and Torrisi in 2001. Three variants considered:
  - algorithm for H-polyhedra (constraint representation);
  - algorithm for V-polyhedra (generator representation);
  - algorithm for VH-polyhedra (double description) in $\mathrm{O}\big(n(l_1 + l_2)m_1 m_2)\big)$.
- A new algorithm for VH-polyhedra:

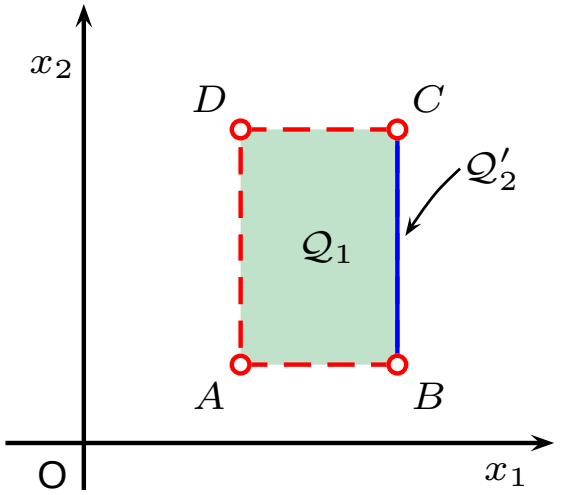
  $\mathcal{P}_1 \uplus \mathcal{P}_2 \neq \mathcal{P}_1 \cup \mathcal{P}_2$ iff $\exists$ constraint β_1 and generator g_1 of $\mathcal{P}_1$ s.t.
  1. g_1 saturates β_1,
  2. $\mathcal{P}_2$ violates β_1, and
  3. $\mathcal{P}_2$ does not subsume g_1.
- (Asymmetric) complexity bound in $\mathrm{O}\big(n(l_1 m_1 + l_1 m_2 + l_2 m_1)\big)$.

Exact Joins for NNC Polyhedra (I)

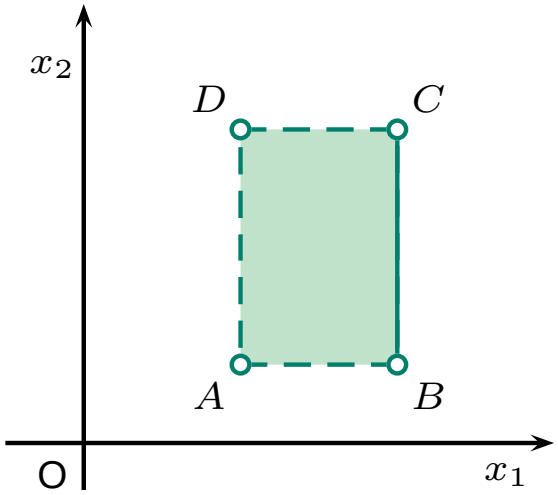
- To our knowledge, the problem has never before been considered.
- Several awkward cases lead to a more complex result. (No technical details here, just a few examples.)



(a) Not Exact

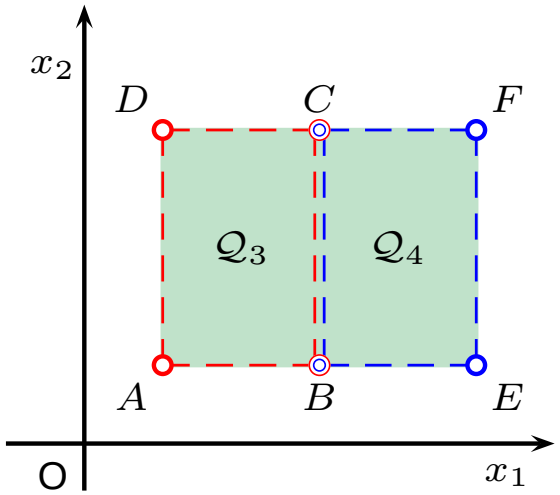


(b) Exact

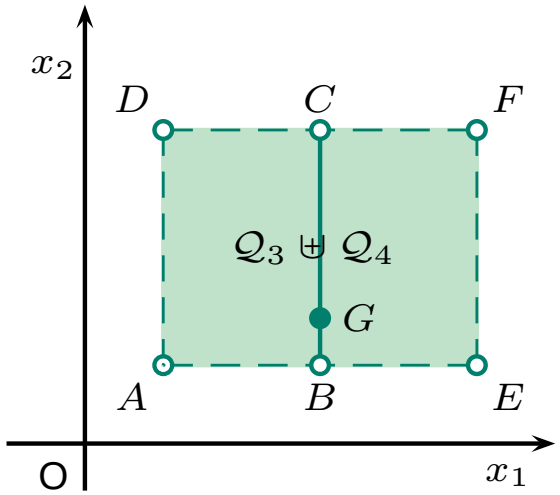


(c) Join

Exact Joins for NNC Polyhedra (II)

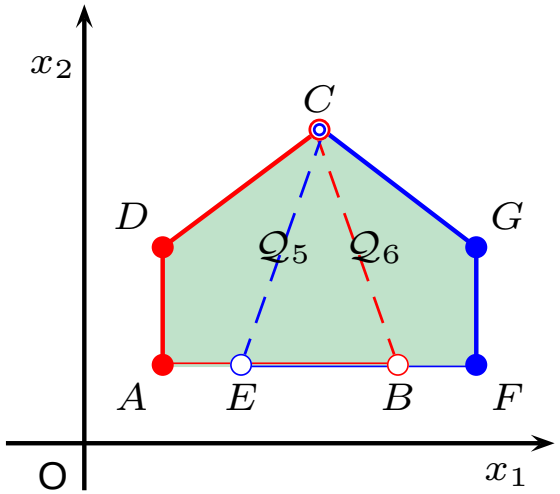


(d) Not Exact

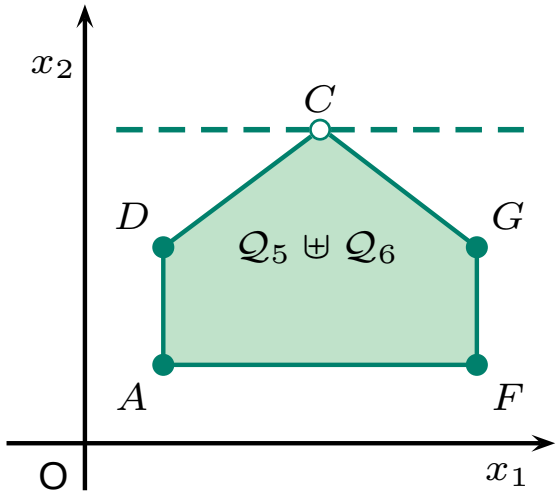


(e) Join

Exact Joins for NNC Polyhedra (III)



(f) Exact

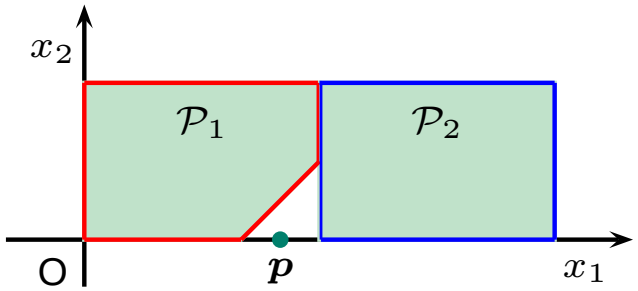


(g) Join

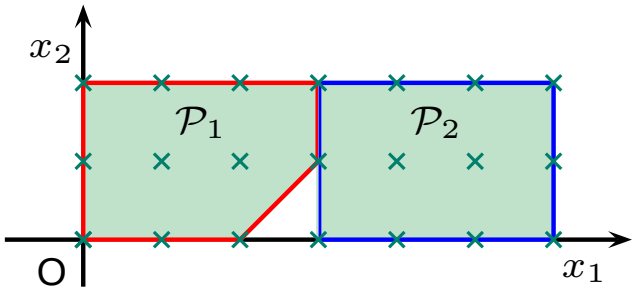
Exact Joins for Other Abstractions

- A domain of convex polyehdra is just one among several possibilities.
- We provide efficient algorithms also for:
  - attribute independent Cartesian products of simple domains such as (rational or integer) intervals, congruence equations, modulo intervals, circular linear progressions;
  - (rational or integer) BD shapes (Bounded Differences);
  - (rational or integer) octagonal shapes.
- Each algorithm is characterized by a worst-case complexity matching the intrinsic complexity of the underlying domain.

EXAMPLE: EXACT JOIN FOR BD SHAPES



(h) Rational Case: Not Exact



(i) Integer Case: Exact

Efficient Approximation of Bounded Arithmetic

- Suppose we want to approximate C/C++ unsigned integers, Java integers, or machine code arithmetic.
- For a relational approach we could use the method of Simon and King.
- An alternative is to define a base, non-relational domain, and then to enrich it with relational information:
  - by combining it with Karr's domain of affine spaces; or
  - by building bounded differences on top of it, i.e., systems of constraints of the form $x_i - x_j \in d_{ij}$, where the d_{ij}'s are elements of the base, non-relational domain.

Circular Delta Sequences (CDS)

- are for approximating w-bits signed or unsigned integer quantities;
- with more precision than both strided intervals (Reps et al.) and circular linear progressions (Sen and Srikant);
- with low complexity operations;

- represented using 3 w-bits numbers:

$$\{ s + \delta * k \}_w^{\mathrm{u}}$$

What Is a CDS?

$$\left\{s + \delta * k\right\}_w^{\mathrm{u}} = \left\{s, (s+\delta) \bmod 2^w, (s+2\delta) \bmod 2^w, \ldots, (s+k\delta) \bmod 2^w\right\}$$

What Is a CDS?

$$\left\{s + \delta * k\right\}_w^{\mathrm{u}} = \left\{s, (s+\delta) \bmod 2^w, (s+2\delta) \bmod 2^w, \ldots, (s+k\delta) \bmod 2^w\right\}$$

For example:

$$\left\{3 + 5 * 4\right\}_3^{\mathrm{u}} = \left\{3, (3+5) \bmod 8, (3+10) \bmod 8, \ldots, (3+20) \bmod 8\right\}$$

What Is a CDS?

$$\{s + \delta * k\}_w^{\mathrm{u}} = \{s, (s+\delta) \bmod 2^w, (s+2\delta) \bmod 2^w, \ldots, (s+k\delta) \bmod$$

For example:

$$\begin{aligned}\{3 + 5 * 4\}_3^{\mathrm{u}} &= \{3, (3+5) \bmod 8, (3+10) \bmod 8, \ldots, (3+20) \bmod 8\} \\ &= \{3,\ 0,\ 5,\ 2,\ 7\}\end{aligned}$$

What Is a CDS?

$$\left\{s + \delta * k\right\}_w^{\mathrm{u}} = \left\{s, (s+\delta) \bmod 2^w, (s+2\delta) \bmod 2^w, \ldots, (s+k\delta) \bmod 2^w\right\}$$

For example:

$$\begin{aligned}\left\{3 + 5 * 4\right\}_3^{\mathrm{u}} &= \left\{3, (3+5) \bmod 8, (3+10) \bmod 8, \ldots, (3+20) \bmod 8\right\} \\ &= \left\{3,\ 0,\ 5,\ 2,\ 7\right\} \\ &= \left\{0,\ 2,\ 3,\ 5,\ 7\right\}\end{aligned}$$

The CDS as an Abstract Domain

- Each cds has a unique normalized representation:
  if $\left\{s + \delta * k\right\}_w^u$ is normalized, then, letting $M = 2^w$,

$$1 \leq \delta \leq M/2, \qquad k \leq M/\gcd(\delta, M) - 1;$$

- many operations on cds's are based on the (extended) Euclidean algorithm for finding the greatest common denominator (optimized to exploit the modulus having the form 2^w), such as:
  - computing the upper bound and lower bound of a cds,
  - checking if a value is a member of a cds,
  - computing the meet and join of two cds's.
- For any given cds, it is cheap to compute the best approximating circular linear progression;
- work is on-going for both the theory of the CDS domain and its efficient implementation.
- This work is joint with Abramo Bagnara and Alessandro Zaccagnini.

Conclusion

- ➜ Numerical domains are very important in the field of the analysis and verification of analog and digital systems.
- ➜ In this field, the complexity/precision tradeoff is particularly severe:
  - on the one hand, giving up precision (e.g., by approximating a polyhedron with a larger one) is allowed and is often necessary to be able to complete the analysis/verification task;
  - on the other hand, giving up too much precision can often prevent the completion of the analysis/verification task.
- ➜ This is why it is important to have a variety of abstract domains and approximate algorithms: even a single application can use a number of them at the same time.
- ➜ Many things remain to be done. . .