Abstract Interpretation (an introduction)

SCSSE Summer School 2017

Part 2

David Pichardie

ENS Rennes, France

these slides are available at
`http://www.irisa.fr/celtique/pichardie/teaching/ecnu-2017-part2.pdf`

Outline

Outline

Outline

Outline

Outline

Non-relational interval environment abstraction

We abstract each variable independently[1] by an interval.

$$\mathrm{Int} \overset{\text{def}}{=} \{\, [a,b] \mid a,b \in \overline{\mathbb{Z}},\ a \leqslant b \,\} \cup \{\bot\} \quad \text{with } \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$$

Examples : $[0,1], [0,+\infty], [-\infty,+\infty], \bot, \cdots$

An abstract environment is hence a mapping from program variable to interval.

$$Env^\sharp \overset{\text{def}}{=} \mathbb{V} \to \mathrm{Int}$$

Example : $[x : [0,1]; y : [0,+\infty]]$

1. See the end of the lecture for a relational abstraction.

Abstract (partial) order

We define interval order by case analysis

$$\frac{I \in \mathrm{Int}}{\bot \sqsubseteq_{\mathrm{Int}} I} \qquad \frac{c \leqslant a \qquad b \leqslant d \qquad a,b,c,d \in \overline{\mathbb{Z}}}{[a,b] \sqsubseteq_{\mathrm{Int}} [c,d]}$$

Abstract environment are pointwise ordered

$$\forall \rho_1^\sharp, \rho_2^\sharp \in Env^\sharp, \quad \rho_1^\sharp \sqsubseteq_{Env}^\sharp \rho_2^\sharp \overset{\text{def}}{=} \forall x \in \mathbb{V},\ \rho_1^\sharp(x) \sqsubseteq_{\mathrm{Int}}^\sharp \rho_2^\sharp(x)$$

Example

$$[x : [0,1];\ y : [0,+\infty]] \sqsubseteq_{\mathrm{Int}}^\sharp [x : [0,2];\ y : [-10,+\infty]]$$

Lattice structure on intervals

Least upper bound

$$\begin{aligned} I \sqcup_{\text{Int}} \bot &\stackrel{\text{def}}{=} I,\ \forall I \in \text{Int} \\ \bot \sqcup_{\text{Int}} I &\stackrel{\text{def}}{=} I,\ \forall I \in \text{Int} \\ [a,b] \sqcup_{\text{Int}} [c,d] &\stackrel{\text{def}}{=} [\min(a,c), \max(b,d)] \end{aligned}$$

Greatest lower bound

$$\begin{aligned} I \sqcap_{\text{Int}} \bot &\stackrel{\text{def}}{=} \bot,\ \forall I \in \text{Int} \\ \bot \sqcap_{\text{Int}} I &\stackrel{\text{def}}{=} \bot,\ \forall I \in \text{Int} \\ [a,b] \sqcap_{\text{Int}} [c,d] &\stackrel{\text{def}}{=} \rho_{\text{Int}}([\max(a,c), \min(b,d)]) \end{aligned}$$

with $\rho_{\text{Int}} \in (\overline{\mathbb{Z}} \times \overline{\mathbb{Z}}) \to \text{Int}$ defined by

$$\rho_{\text{Int}}(a,b) = \begin{cases} [a,b] & \text{if } a \leqslant b, \\ \bot & \text{otherwise} \end{cases}$$

Exercice

Give an example of infinite ascending chain in Int.

Give an example of infinite descending chain in Int.

(All these examples will have bad impact on fixpoint iterations later in the course...)

Concretization

Intervals are concretized into a property on $\mathbb{Z}$.

$$\begin{array}{rcl}\gamma_{\mathrm{Int}}(\bot) & \stackrel{\mathrm{def}}{=} & \emptyset \\ \gamma_{\mathrm{Int}}([a,b]) & \stackrel{\mathrm{def}}{=} & \{\, z \in \mathbb{Z} \mid a \leqslant z \text{ and } z \leqslant b \,\}\end{array}$$

Abstract environments are concretized into a property on $\mathit{Env} = \mathbb{V} \to \mathbb{Z}$.

$$\begin{array}{l}\forall \rho^\sharp \in \mathit{Env}^\sharp, \\ \qquad \gamma_{\mathit{Env}}(\rho^\sharp) \stackrel{\mathrm{def}}{=} \left\{\, \rho \mid \forall x \in \mathbb{V},\ \rho(x) \in \gamma_{\mathrm{Int}}(\rho^\sharp(x)) \,\right\}\end{array}$$

Programming $[\![x := ?]\!]^\sharp$

To forget x, we assign $\top$ to it

$$[\![x := ?]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \top_{\text{Int}}]$$

with $\top_{\text{Int}} = [-\infty, +\infty]$.

Construction of $[\![x := e]\!]^\sharp$

$$[\![x := e]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \ \ldots\,]$$

Some easy cases :

$$[\![x := n]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \ \ \ldots\ ]$$

$$[\![x := y]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \ \ \ldots\ ]$$

Construction of $[\![x := e]\!]^\sharp$

$$[\![x := e]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \;\ldots\,]$$

Some easy cases :

$$[\![x := n]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \;[n,n]\,]$$

$$[\![x := y]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \;\ldots\,]$$

Construction of $[\![x := e]\!]^\sharp$

$$[\![x := e]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \ \dots\,]$$

Some easy cases :

$$[\![x := n]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp[x \mapsto \ [n, n]\,]$$

$$[\![x := y]\!]^\sharp\,(\rho^\sharp) = \rho^\sharp\big[x \mapsto \ \rho^\sharp(x)\,\big]$$

Construction of $[\![x := e]\!]^\sharp$

General case

$$[\![x := e]\!]^\sharp(\rho^\sharp) = \rho^\sharp\Big[x \mapsto \mathcal{A}[\![e]\!]^\sharp(\rho^\sharp)\Big]$$

with

$$\forall e \in \mathrm{Expr},\ \mathcal{A}[\![e]\!]^\sharp \in Env^\sharp \to \mathrm{Int}$$

a (forward) abstract evaluation of expressions

$$\begin{array}{rcl} \mathcal{A}[\![n]\!]^\sharp(\rho^\sharp) & = & [n, n] \\ \mathcal{A}[\![x]\!]^\sharp(\rho^\sharp) & = & \rho^\sharp(x) \\ \mathcal{A}[\![e_1\, o\, e_2]\!]^\sharp(\rho^\sharp) & = & o^\sharp\left(\mathcal{A}[\![e_1]\!]^\sharp(\rho^\sharp), \mathcal{A}[\![e_2]\!]^\sharp(\rho^\sharp)\right) \end{array}$$

with $o^\sharp \in \mathrm{Int} \times \mathrm{Int} \to \mathrm{Int}$ an abstract arithmetic binary operator.

Abstract binary operators

Addition

$$\begin{aligned} +^\sharp(I, \bot) &= I \\ +^\sharp(\bot, I) &= I \\ +^\sharp([a,b],[c,d]) &= \end{aligned}$$

Soustraction

$$\begin{aligned} -^\sharp(I, \bot) &= I \\ -^\sharp(\bot, I) &= I \\ -^\sharp([a,b],[c,d]) &= \end{aligned}$$

Multiplication

$$\begin{aligned} \times^\sharp(I, \bot) &= I \\ \times^\sharp(\bot, I) &= I \\ \times^\sharp([a,b],[c,d]) &= \end{aligned}$$

Abstract binary operators

Addition

$$\begin{aligned} +^\sharp(I, \bot) &= I \\ +^\sharp(\bot, I) &= I \\ +^\sharp([a,b],[c,d]) &= [a+c, b+d] \end{aligned}$$

Soustraction

$$\begin{aligned} -^\sharp(I, \bot) &= I \\ -^\sharp(\bot, I) &= I \\ -^\sharp([a,b],[c,d]) &= \end{aligned}$$

Multiplication

$$\begin{aligned} \times^\sharp(I, \bot) &= I \\ \times^\sharp(\bot, I) &= I \\ \times^\sharp([a,b],[c,d]) &= \end{aligned}$$

Abstract binary operators

Addition

$$\begin{aligned} +^\sharp(I,\bot) &= I \\ +^\sharp(\bot,I) &= I \\ +^\sharp([a,b],[c,d]) &= [a+c,b+d] \end{aligned}$$

Soustraction

$$\begin{aligned} -^\sharp(I,\bot) &= I \\ -^\sharp(\bot,I) &= I \\ -^\sharp([a,b],[c,d]) &= [a-d,b-c] \end{aligned}$$

Multiplication

$$\begin{aligned} \times^\sharp(I,\bot) &= I \\ \times^\sharp(\bot,I) &= I \\ \times^\sharp([a,b],[c,d]) &= \end{aligned}$$

Abstract binary operators

Addition

$$\begin{aligned} +^\sharp(I,\bot) &= I \\ +^\sharp(\bot,I) &= I \\ +^\sharp([a,b],[c,d]) &= [a+c,b+d] \end{aligned}$$

Soustraction

$$\begin{aligned} -^\sharp(I,\bot) &= I \\ -^\sharp(\bot,I) &= I \\ -^\sharp([a,b],[c,d]) &= [a-d,b-c] \end{aligned}$$

Multiplication

$$\begin{aligned} \times^\sharp(I,\bot) &= I \\ \times^\sharp(\bot,I) &= I \\ \times^\sharp([a,b],[c,d]) &= [\min(ac,ad,bc,bd),\max(ac,ad,bc,bd)] \end{aligned}$$

Example

If $\rho^\sharp(x) = [2, 4]$,

$$\mathcal{A}[\![(2 \times x) + 1]\!]^\sharp(\rho^\sharp) \quad =$$

Example

If $\rho^\sharp(x) = [2, 4]$,

$$\mathcal{A}[\![(2 \times x) + 1]\!]^\sharp(\rho^\sharp) \quad = \quad +^\sharp\left(\times^\sharp([2,2], \rho^\sharp(x)), [1,1]\right)$$

Example

If $\rho^\sharp(x) = [2,4]$,

$$\begin{array}{rcl} \mathcal{A}[\![(2 \times x) + 1]\!]^\sharp(\rho^\sharp) & = & +^\sharp\left(\times^\sharp([2,2], \rho^\sharp(x)), [1,1]\right) \\ & = & +^\sharp\left(\times^\sharp([2,2], [2,4]), [1,1]\right) \end{array}$$

Example

If $\rho^\sharp(x) = [2,4]$,

$$\begin{aligned}
\mathcal{A}[\![(2 \times x) + 1]\!]^\sharp(\rho^\sharp) &= +^\sharp\left(\times^\sharp([2,2], \rho^\sharp(x)), [1,1]\right) \\
&= +^\sharp\left(\times^\sharp([2,2], [2,4]), [1,1]\right) \\
&= +^\sharp([4,8], [1,1])
\end{aligned}$$

Example

If $\rho^\sharp(x) = [2, 4]$,

$$\begin{aligned}
\mathcal{A}[\![(2 \times x) + 1]\!]^\sharp(\rho^\sharp) &= +^\sharp\left(\times^\sharp([2,2], \rho^\sharp(x)), [1,1]\right) \\
&= +^\sharp\left(\times^\sharp([2,2], [2,4]), [1,1]\right) \\
&= +^\sharp([4,8], [1,1]) \\
&= [5,9]
\end{aligned}$$

Construction of $[\![t]\!]^\sharp$

More difficult, because ideally such a refinement should be possible...

$$[\![x + y > 0]\!]^\sharp\,([x \mapsto [-2,4];\ y \mapsto [2,4]]) = ([x \mapsto [-2,4];\ y \mapsto [2,4]])$$

Construction of $[\![t]\!]^\sharp$

$$\begin{array}{rcl}[\![e_1\ c\ e_2]\!]^\sharp(\rho^\sharp) & = & \left([\![e_1]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n_1^\sharp) \sqcap^\sharp_{Env} [\![e_2]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n_2^\sharp)\right) \\ & & \text{with } (n_1^\sharp, n_2^\sharp) = [\![c]\!]\downarrow^\sharp_{\text{comp}}\left(\mathcal{A}[\![e_1]\!]^\sharp(\rho^\sharp), \mathcal{A}[\![e_2]\!]^\sharp(\rho^\sharp)\right)\end{array}$$

- $[\![c]\!]\downarrow^\sharp_{\text{comp}} \in \text{Int}^\sharp \times \text{Int}^\sharp \to \text{Int}^\sharp \times \text{Int}^\sharp$ computes a refinement of two numeric abstract values, knowing that they verify condition c
- $[\![e]\!]\downarrow^\sharp_{\text{expr}} \in Env^\sharp \times \text{Int}^\sharp \to Env^\sharp$: $[\![e]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n^\sharp)$ computes a refinement of the abstract environment $\rho^\sharp$, knowing that the expression e evaluates into a value that is approximated by $n^\sharp$ in this environment.

Required operators on the numeric abstraction

$$
\begin{array}{rcl}
[\![c]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n^\sharp) & = & \begin{cases} \bot_{Env} & \text{if } [c, c] \sqcap^\sharp_{\text{Int}} n^\sharp = \bot_{\text{Int}} \\ \rho^\sharp & \text{otherwise} \end{cases} \\
[\![x]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n^\sharp) & = & (\rho^\sharp[x \mapsto \rho^\sharp(x) \sqcap^\sharp_{\text{Int}} n^\sharp]) \\
[\![e_1\, o\, e_2]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n^\sharp) & = & \left([\![e_1]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n_1^\sharp) \sqcap^\sharp_{Env} [\![e_2]\!]\downarrow^\sharp_{\text{expr}}(\rho^\sharp, n_2^\sharp)\right) \\
\multicolumn{3}{r}{\text{with } (n_1^\sharp, n_2^\sharp) = [\![o]\!]\downarrow^\sharp_{\text{op}}(n^\sharp, \mathcal{A}[\![e_1]\!]^\sharp(\rho^\sharp), \mathcal{A}[\![e_2]\!]^\sharp(\rho^\sharp))}
\end{array}
$$

Required operators on the numeric abstraction

$$[\![o]\!]\downarrow^{\sharp}_{\text{op}} \in \text{Int}^{\sharp} \times \text{Int}^{\sharp} \times \text{Int}^{\sharp} \to \text{Int}^{\sharp} \times \text{Int}^{\sharp}$$

$[\![o]\!]\downarrow^{\sharp}_{\text{op}}\ (n^{\sharp}, n^{\sharp}_1, n^{\sharp}_2)$ computes a refinement of two numeric values $n^{\sharp}_1$ and $n^{\sharp}_2$ knowing that the result of the binary operation o is approximated by $n^{\sharp}$ on their concretisations.

Backward abstractions on intervals (1/2)

All these operators are *stricts* : they return $\perp$ if one of their arguments is $\perp$.

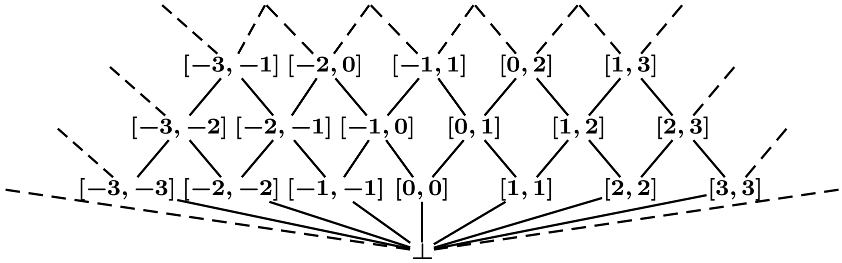
$$\begin{array}{rcl}
[\![=]\!]\!\downarrow^{\sharp}_{\text{comp}}\,([a,b],[c,d]) & = & ([a,b]\sqcap_{\text{Int}}[c,d],[a,b]\sqcap_{\text{Int}}[c,d]) \\
[\![<]\!]\!\downarrow^{\sharp}_{\text{comp}}\,([a,b],[c,d]) & = & ([a,b]\sqcap_{\text{Int}}[-\infty,d-1],[a+1,+\infty]\sqcap_{\text{Int}}[c,d]) \\
[\![\leqslant]\!]\!\downarrow^{\sharp}_{\text{comp}}\,([a,b],[c,d]) & = & ([a,b]\sqcap_{\text{Int}}[-\infty,d],[a,+\infty]\sqcap_{\text{Int}}[c,d]) \\
[\![\neq]\!]\!\downarrow^{\sharp}_{\text{comp}}\,([a,b],[c,d]) & = & ?\ \textit{exercise...}
\end{array}$$

Backward abstractions on intervals (1/2)

All these operators are *stricts* : they return $\perp$ if one of their arguments is $\perp$.

$$
\begin{array}{rcl}
[\![+]\!]\!\downarrow^\sharp_{\mathrm{op}}([a,b],[c,d],[e,f]) & = & (\rho(\max(c,a-f),\min(d,b-e)),\\
 & & \quad \rho(\max(e,a-d),\min(f,b-c)))\\
[\![-]\!]\!\downarrow^\sharp_{\mathrm{op}}([a,b],[c,d],[e,f]) & = & (\rho(\max(c,a+e),\min(d,b+f)),\\
 & & \rho(\max(e,c-b),\min(f,d-a)))\\
[\![*]\!]\!\downarrow^\sharp_{\mathrm{op}}([a,b],[c,d],[e,f]) & = & ([c,d],[e,f])
\end{array}
$$

Convergence problem



Such a lattice does not satisfy the ascending chain condition.

Example of infinite increasing chain :

$$\perp \sqsubset [0,0] \sqsubset [0,1] \sqsubset \cdots \sqsubset [0,n] \sqsubset \cdots$$

Solution : dynamic approximation

- we extrapolate the limit thanks to a widening operator ∇

$$\perp \sqsubset [0,0] \sqsubset [0,1] \sqsubset [0,2] \sqsubset [0,+\infty] = [0,2]\nabla[0,3]$$

Outline

Fixpoint approximation

Lemma

Let $(A, \sqsubseteq, \sqcup, \sqcap)$ a complete lattice and f a monotone operator on A. If a is a post-fixpoint of f (i.e. $f(a) \sqsubseteq a$), then $\mathrm{lfp}(f) \sqsubseteq a$.

We may want to compute an over-approximation of $\mathrm{lfp}(f)$ in the following cases :

- The lattice does not satisfies the ascending chain condition, the iteration $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ may never terminates.
- The ascending chain condition is satisfied but the iteration chain is too long to allow an efficient computation.
- If the underlying lattice is not complete, the limits of the ascending iterations do not necessarily belongs to the abstraction domain.

Widening

Idea : the standard iteration is of the form

$$x^0 = \bot, x^{n+1} = F(x^n) \quad = x^n \sqcup F(x^n)$$

We will replace it by something of the form

$$y^0 = \bot, y^{n+1} = y^n \nabla F(y^n)$$

such that

(i) (y^n) is increasing,

(ii) $x^n \sqsubseteq y^n$, for all n,

(iii) and (y^n) stabilizes after a finite number of steps.

But we also want a ∇ operator that is independent of F.

Widening : definition

A widening is an operator $\nabla : L \times L \to L$ such that

- $\forall x, x' \in L, x \sqcup x' \sqsubseteq x \nabla x'$ (implies (i) & (ii))
- If $x^0 \sqsubseteq x^1 \sqsubseteq \dots$ is an increasing chain, then the increasing chain $y^0 = x^0, y^{n+1} = y^n \nabla x^{n+1}$ stabilizes after a finite number of steps (implies (iii)).

Usage : we replace $x^0 = \bot, x^{n+1} = F(x^n)$
by $y^0 = \bot, y^{n+1} = y^n \nabla F(y^n)$

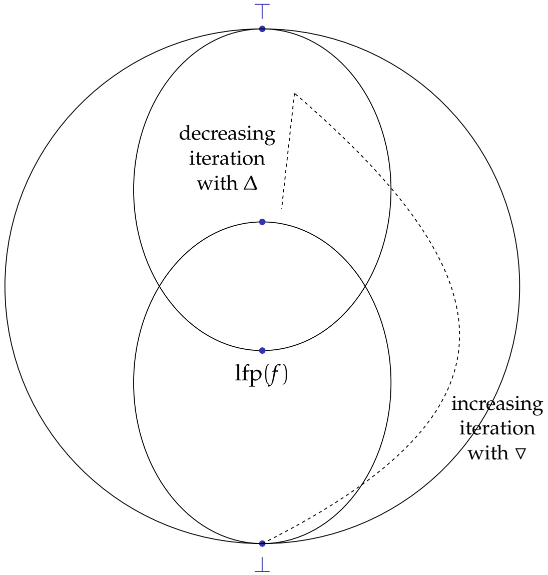
Widening : theorem

Theorem

Let L a complete lattice, $F : L \to L$ a monotone function and $\nabla : L \times L \to L$ a widening operator. The chain $y^0 = \bot, y^{n+1} = y^n \nabla F(y^n)$ stabilizes after a finite number of steps towards a post-fixpoint y of F.

Corollary : $\mathrm{lfp}(F) \sqsubseteq y$.

Scheme


$\top$
decreasing iteration with Δ
$\mathrm{lfp}(f)$
increasing iteration with ∇
$\bot$

Example : widening on intervals

Idea : as soon as a bound is not stable, we extrapolate it by $+\infty$ (or $-\infty$). After such an extrapolation, the bound can't move any more.

Definition :

$$\begin{array}{lcl} [a,b]\nabla_{\text{Int}}[a',b'] & = & [\ \ \text{if } a' < a \text{ then } -\infty \text{ else } a, \\ & & \text{if } b' > b \text{ then } +\infty \text{ else } b\,] \\ \perp\nabla_{\text{Int}}[a',b'] & = & [a',b'] \\ I\ \nabla_{\text{Int}}\ \perp & = & I \end{array}$$

Examples :
$[-3,4]\nabla_{\text{Int}}[-3,2] = [-3,4]$
$[-3,4]\nabla_{\text{Int}}[-3,5] = [-3,+\infty]$

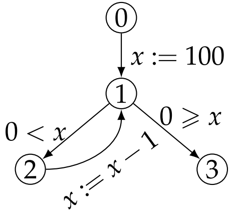
Example

```
x := 100;

while 0 < x {

  x := x − 1;
}
```

$$X_1 = [100, 100] \sqcup_{\mathrm{Int}} \left(X_2 -^\sharp [1, 1]\right)$$

$$X_2 = [1, +\infty] \sqcap_{\mathrm{Int}} X_1$$

$$X_3 = [-\infty, 0] \sqcap_{\mathrm{Int}} X_1$$



Example : without widening

$$\begin{aligned} X_1 &= [100,100] \sqcup_{\text{Int}} \left(X_2 -^\sharp [1,1]\right) \\ X_2 &= [1,+\infty] \sqcap_{\text{Int}} X_1 \\ X_3 &= [-\infty,0] \sqcap_{\text{Int}} X_1 \end{aligned}$$

Iteration strategy : $1 \to 2 \to 3 \to 1 \to 2 \to \cdots$

$$\begin{array}{ll} X_1^0 = \bot & X_1^{n+1} = [100,100] \sqcup_{\text{Int}} \left(X_2^n -^\sharp [1,1]\right) \\ X_2^0 = \bot & X_2^{n+1} = [1,+\infty] \sqcap_{\text{Int}} X_1^{n+1} \\ X_3^0 = \bot & X_3^{n+1} = [-\infty,0] \sqcap_{\text{Int}} X_1^{n+1} \end{array}$$

X_1	$\bot$	$\cdots$
X_2	$\bot$	$\cdots$
X_3	$\bot$	$\cdots$

Example : without widening

$$\begin{aligned} X_1 &= [100,100] \sqcup_{\text{Int}} \left(X_2 -^\sharp [1,1]\right) \\ X_2 &= [1,+\infty] \sqcap_{\text{Int}} X_1 \\ X_3 &= [-\infty,0] \sqcap_{\text{Int}} X_1 \end{aligned}$$

Iteration strategy : $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \cdots$

$$\begin{array}{ll} X_1^0 = \bot & X_1^{n+1} = [100,100] \sqcup_{\text{Int}} \left(X_2^n -^\sharp [1,1]\right) \\ X_2^0 = \bot & X_2^{n+1} = [1,+\infty] \sqcap_{\text{Int}} X_1^{n+1} \\ X_3^0 = \bot & X_3^{n+1} = [-\infty,0] \sqcap_{\text{Int}} X_1^{n+1} \end{array}$$

X_1	$\bot$	$[100,100]$	$[99,100]$	$[98,100]$	$[97,100]$	$\cdots$	$[1,100]$	$[0,100]$
X_2	$\bot$	$[100,100]$	$[99,100]$	$[98,100]$	$[97,100]$	$\cdots$	$[1,100]$	$[1,100]$
X_3	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\cdots$	$\bot$	$[0,0]$

Example : with widening at each nodes of the cfg

$$\begin{aligned} X_1 &= [100,100] \sqcup_{\text{Int}} \left(X_2 -^\sharp [1,1]\right) \\ X_2 &= [1,+\infty] \sqcap_{\text{Int}} X_1 \\ X_3 &= [-\infty,0] \sqcap_{\text{Int}} X_1 \end{aligned}$$

Iteration strategy : $1 \to 2 \to 3 \to 1 \to 2 \to \cdots$

$$\begin{array}{ll} X_1^0 = \bot & X_1^{n+1} = X_1^n \nabla_{\text{Int}} \left([100,100] \sqcup_{\text{Int}} \left(X_2^n -^\sharp [1,1]\right)\right) \\ X_2^0 = \bot & X_2^{n+1} = X_2^n \nabla_{\text{Int}} \left([1,+\infty] \sqcap_{\text{Int}} X_1^{n+1}\right) \\ X_3^0 = \bot & X_3^{n+1} = X_3^n \nabla_{\text{Int}} \left([-\infty,0] \sqcap_{\text{Int}} X_1^{n+1}\right) \end{array}$$

X_1	$\bot$
X_2	$\bot$
X_3	$\bot$

Example : with widening at each nodes of the cfg

$$\begin{aligned}
X_1 &= [100,100] \sqcup_{\text{Int}} \left(X_2 -^\sharp [1,1]\right)\\
X_2 &= [1,+\infty] \sqcap_{\text{Int}} X_1\\
X_3 &= [-\infty,0] \sqcap_{\text{Int}} X_1
\end{aligned}$$

Iteration strategy : $1 \to 2 \to 3 \to 1 \to 2 \to \cdots$

$$\begin{array}{ll}
X_1^0 = \bot & X_1^{n+1} = X_1^n \nabla_{\text{Int}} \left([100,100] \sqcup_{\text{Int}} \left(X_2^n -^\sharp [1,1]\right)\right)\\
X_2^0 = \bot & X_2^{n+1} = X_2^n \nabla_{\text{Int}} \left([1,+\infty] \sqcap_{\text{Int}} X_1^{n+1}\right)\\
X_3^0 = \bot & X_3^{n+1} = X_3^n \nabla_{\text{Int}} \left([-\infty,0] \sqcap_{\text{Int}} X_1^{n+1}\right)
\end{array}$$

X_1	$\bot$	$[100,100]$	$[-\infty,100]$
X_2	$\bot$	$[100,100]$	$[-\infty,100]$
X_3	$\bot$	$\bot$	$[-\infty,0]$

Improving fixpoint approximation

Idea : iterating a little more may help...

Theorem

Let $(A, \sqsubseteq, \sqcup, \sqcap)$ *a complete lattice,* f *a monotone operator on* A *and* a *a post-fixpoint of* f*. The chain* $(x_n)_n$ *defined by* $\left\{ \begin{array}{lcl} x_0 & = & a \\ x_{k+1} & = & f(x_k) \end{array} \right.$ *admits for limit* $(\bigsqcup\{x_n\})$ *the greatest fixpoint of* f *lower than* a *(written* $\mathrm{gfp}_a(f)$*). In particular,* $\mathrm{lfp}(f) \sqsubseteq \bigsqcup\{x_n\}$*.*
Each intermediate step is a correct approximation :

$$\forall k,\ \mathrm{lfp}(f) \sqsubseteq \mathrm{gfp}_a(f) \sqsubseteq x_k \sqsubseteq a$$

Narrowing : definition

A *narrowing* is an operator $\Delta : L \times L \to L$ such that

- $\forall x, x' \in L, x' \sqsubseteq x \Delta x' \sqsubseteq x$
- If $x^0 \sqsupseteq x^1 \sqsupseteq \dots$ is a decreasing chain, then the increasing chain $y^0 = x^0, y^{n+1} = y^n \Delta x^{n+1}$ stabilizes after a finite number of steps.

Narrowing : decreasing iteration

Theorem

If Δ is a narrowing operator on a poset $(A, \sqsubseteq)$, if f is a monotone operator on A and a is a post-fixpoint of f then the chain $(x_n)_n$ defined by $\left\{\begin{array}{lcl} x_0 & = & a \\ x_{k+1} & = & x_k \Delta f(x_k) \end{array}\right.$ *stabilizes after a finite number of steps on a post-fixpoint of f lower than a.*

Narrowing on intervals

$$\begin{array}{rcl}
[a,b]\Delta_{\text{Int}}[c,d] & = & [\text{if } a=-\infty \text{ then } c \text{ else } a \;\; ; \;\; \text{if } b=+\infty \text{ then } d \text{ else } b] \\
I \;\; \Delta_{\text{Int}} \;\; \bot & = & \bot \\
\bot \;\; \Delta_{\text{Int}} \;\; I & = & \bot
\end{array}$$

Intuition : we only improve infinite bounds.

In practice : a few standard iterations already improve a lot the result that has been obtained after widening...

- Assignments by constants and conditional guards make the decreasing iterations efficient : they *filter* the (too big) approximations computed by the widening

Example : with narrowing at each nodes of the cfg

$$\begin{aligned}
X_1 &= [100,100] \sqcup_{\text{Int}} \left(X_2 -^\sharp [1,1]\right)\\
X_2 &= [1,+\infty] \sqcap_{\text{Int}} X_1\\
X_3 &= [-\infty,0] \sqcap_{\text{Int}} X_1
\end{aligned}$$

Iteration strategy : $1 \to 2 \to 3 \to 1 \to 2 \to \cdots$

$$\begin{array}{ll}
X_1^0 = [-\infty,100] & X_1^{n+1} = X_1^n \Delta_{\text{Int}} \left([100,100] \sqcup_{\text{Int}} \left(X_2^n -^\sharp [1,1]\right)\right)\\
X_2^0 = [-\infty,100] & X_2^{n+1} = X_2^n \Delta_{\text{Int}} \left([1,+\infty] \sqcap_{\text{Int}} X_1^{n+1}\right)\\
X_3^0 = [-\infty,0] & X_3^{n+1} = X_3^n \Delta_{\text{Int}} \left([-\infty,0] \sqcap_{\text{Int}} X_1^{n+1}\right)
\end{array}$$

X_1	$[-\infty,100]$
X_2	$[-\infty,100]$
X_3	$[-\infty,0]$

Example : with narrowing at each nodes of the cfg

$$\begin{aligned} X_1 &= [100,100] \sqcup_{\mathrm{Int}} \left(X_2 -^{\sharp} [1,1]\right) \\ X_2 &= [1,+\infty] \sqcap_{\mathrm{Int}} X_1 \\ X_3 &= [-\infty,0] \sqcap_{\mathrm{Int}} X_1 \end{aligned}$$

Iteration strategy : $1 \to 2 \to 3 \to 1 \to 2 \to \cdots$

$$\begin{array}{ll} X_1^0 = [-\infty,100] & X_1^{n+1} = X_1^n \Delta_{\mathrm{Int}} \left([100,100] \sqcup_{\mathrm{Int}} \left(X_2^n -^{\sharp} [1,1]\right)\right) \\ X_2^0 = [-\infty,100] & X_2^{n+1} = X_2^n \Delta_{\mathrm{Int}} \left([1,+\infty] \sqcap_{\mathrm{Int}} X_1^{n+1}\right) \\ X_3^0 = [-\infty,0] & X_3^{n+1} = X_3^n \Delta_{\mathrm{Int}} \left([-\infty,0] \sqcap_{\mathrm{Int}} X_1^{n+1}\right) \end{array}$$

X_1	$[-\infty,100]$	$[-\infty,100]$	$[0,100]$
X_2	$[-\infty,100]$	$[1,100]$	$[1,100]$
X_3	$[-\infty,0]$	$[-\infty,0]$	$[0,0]$

The particular case of an equation system

Consider a system

$$\left\{\begin{array}{lcl} x_1 & = & f_1(x_1,\ldots,x_n) \\ & \vdots & \\ x_n & = & f_n(x_1,\ldots,x_n) \end{array}\right.$$

with $f_1,\ldots,f_n$ monotones.

Standard iteration :

$$\begin{array}{lcl} x_1^{i+1} & = & f_1(x_1^i,\ldots,x_n^i) \\ x_2^{i+1} & = & f_2(x_1^i,\ldots,x_n^i) \\ & \vdots & \\ x_n^{i+1} & = & f_n(x_1^i,\ldots,x_n^i) \end{array}$$

Standard iteration with widening :

$$\begin{array}{lcl} x_1^{i+1} & = & x_1^i \nabla f_1(x_1^i,\ldots,x_n^i) \\ x_2^{i+1} & = & x_2^i \nabla f_2(x_1^i,\ldots,x_n^i) \\ & \vdots & \\ x_n^{i+1} & = & x_n^i \nabla f_n(x_1^i,\ldots,x_n^i) \end{array}$$

The particular case of an equation system

$$\begin{cases} x_1 & = & f_1(x_1,\ldots,x_n) \\ & \vdots & \\ x_n & = & f_n(x_1,\ldots,x_n) \end{cases}$$

It is sufficient (and generally more precise) to use ∇ for a selection of index W such that each dependence cycle in the system goes through at least one point in W.

$$\forall k = 1..n,\ x_k^{i+1} = \begin{array}{ll} x_k^i \nabla f_k(x_1^i,\ldots,x_n^i) & \text{if } k \in W \\ f_k(x_1^i,\ldots,x_n^i) & \text{otherwise} \end{array}$$

Chaotic iteration : at each step, we use only one equation, without forgetting one for ever.

Contrary, to what happen in a standard dataflow framework (with monotone functions and ascending chain condition), the iteration strategy may affect a lot the precision of the result. See F. Bourdoncle, *Efficient Chaotic Iteration Strategies with Widenings*, 1993.

Outline

Dealing with pointers

We now extend our language with memory operations.

$$\begin{array}{rcll}
Exp & ::= & n & n \in \mathbb{Z} \\
& \mid & ? & \\
& \mid & x & x \in \mathbb{V} \\
& \mid & Exp\ o\ Exp & o \in \{+, -, \times\} \\
& & & \\
& \mid & \&X & X \in \mathbb{B}\ \text{(statically allocated} \\
& & & \qquad\qquad \text{memory blocks)} \\
& \mid & *Exp & \text{(memory read)}
\end{array}$$

Dealing with pointers

We now extend our language with memory operations.

$$
\begin{array}{rll}
\mathit{Stm} ::= & {}^{l}[x \ := \ \mathit{Exp}] & \quad l \in \mathbb{P} \\
| & {}^{l}[\texttt{skip}] & \\
| & \texttt{if}\ {}^{l}[\mathit{Test}]\ \{\ \mathit{Stm}\ \}\ \{\ \mathit{Stm}\ \} & \\
| & \texttt{while}\ {}^{l}[\mathit{Test}]\ \{\ \mathit{Stm}\ \} & \\
| & \mathit{Stm}\ ;\ \mathit{Stm} & \\
& & \\
| & {}^{l}[*\mathit{Exp} := \mathit{Exp}] & \quad \text{(memory write)}
\end{array}
$$

While semantics (updated)

Semantic domains

$$\begin{array}{rcl} \mathit{Val} & ::= & \mathsf{Num}(n) \;\; n \in \mathbb{Z} \\ & | & \mathsf{Ptr}(b,n) \;\; b \in \mathbb{B},\ n \in \mathbb{Z} \\ \mathit{Env} & \stackrel{\text{def}}{=} & \mathbb{V} \to \mathit{Val} \\ \mathit{Mem} & \stackrel{\text{def}}{=} & (\mathbb{B} \times \mathbb{N}) \to \mathit{Val} \\ \mathit{State} & \stackrel{\text{def}}{=} & \mathbb{P} \times \mathit{Env} \times \mathit{Mem} \end{array}$$

We assume each block $b \in \mathbb{B}$ as a size $\mathit{size}(b) \in \mathbb{N}$.

For each memory $\sigma \in \mathit{Mem}$, we only take care of the values $\sigma(b,i)$ for $0 \leqslant i < \mathit{size}(b)$. Reading or writing other locations leads to a semantic error.

Semantics of expressions (updated)

$$\begin{array}{lcl}
\mathcal{A}[\![n]\!]\,\rho,\sigma & = & \{\,n\,\} \\
\mathcal{A}[\![?]\!]\,\rho,\sigma & = & \mathbb{Z} \\
\mathcal{A}[\![x]\!]\,\rho,\sigma & = & \{\,\rho(x)\,\},\ x \in \mathbb{V} \\
\mathcal{A}[\![e_1\ \times\ e_2]\!]\,\rho,\sigma & = & \{\,\mathsf{Num}(n_1 \times n_2) \mid \mathsf{Num}(n_1) \in \mathcal{A}[\![e_1]\!]\,\rho,\sigma,\ \mathsf{Num}(n_2) \in \mathcal{A}[\![e_2]\!]\,\rho,\sigma\,\} \\
\mathcal{A}[\![e_1\ -\ e_2]\!]\,\rho,\sigma & = & \{\,\mathsf{Num}(n_1 - n_2) \mid \mathsf{Num}(n_1) \in \mathcal{A}[\![e_1]\!]\,\rho,\sigma,\ \mathsf{Num}(n_2) \in \mathcal{A}[\![e_2]\!]\,\rho,\sigma\,\} \\
 & & \cup\,\{\,\mathsf{Ptr}(b, n_1 - n_2) \mid \mathsf{Ptr}(b, n_1) \in \mathcal{A}[\![e_1]\!]\,\rho,\sigma,\ \mathsf{Num}(n_2) \in \mathcal{A}[\![e_2]\!]\,\rho,\sigma\,\} \\
\mathcal{A}[\![e_1\ +\ e_2]\!]\,\rho,\sigma & = & \{\,\mathsf{Num}(n_1 + n_2) \mid \mathsf{Num}(n_1) \in \mathcal{A}[\![e_1]\!]\,\rho,\sigma,\ \mathsf{Num}(n_2) \in \mathcal{A}[\![e_2]\!]\,\rho,\sigma\,\} \\
 & & \cup\,\{\,\mathsf{Ptr}(b, n_1 + n_2) \mid \mathsf{Ptr}(b, n_1) \in \mathcal{A}[\![e_1]\!]\,\rho,\sigma,\ \mathsf{Num}(n_2) \in \mathcal{A}[\![e_2]\!]\,\rho,\sigma\,\} \\
\mathcal{A}[\![\&X]\!]\,\rho,\sigma & = & \{\,\mathsf{Ptr}(X, 0)\} \\
\mathcal{A}[\![*e]\!]\,\rho,\sigma & = & \{\,\sigma(b,i) \mid \mathsf{Ptr}(b,i) \in \mathcal{A}[\![e]\!]\,\rho,\sigma,\ 0 \leqslant i < \mathit{size}(b)\}
\end{array}$$

Structural Operational Semantics (updated)

$$\frac{v \in \mathcal{A}[\![a]\!]\rho, \sigma}{({}^{l}[x := a], \rho, \sigma) \Rightarrow \rho[x \mapsto v], \sigma}$$

$$\ldots$$

$$\frac{(S_1, \rho, \sigma) \Rightarrow (S_1', \rho', \sigma')}{(S_1 \,;\, S_2, \rho, \sigma) \Rightarrow (S_1' \,;\, S_2, \rho', \sigma')}$$

$$\ldots$$

$$\frac{\mathsf{Ptr}(b, i) \in \mathcal{A}[\![e]\!]\rho, \sigma \qquad 0 \leqslant i < size(b) \qquad v \in \mathcal{A}[\![a]\!]\rho, \sigma}{({}^{l}[*e := a], \rho, \sigma) \Rightarrow \rho, \sigma[(b, i) \mapsto v]}$$

Memory abstraction

Since we consider statically allocated and fixed-size memory blocks, we can [2] statically know the domain of the memory.
$\Rightarrow$ we abstract each memory cell with an abstract value.

$$Mem^\sharp \stackrel{\text{def}}{=} (\mathbb{B} \times \mathbb{N}) \rightarrow Val^\sharp$$

$$\gamma(\sigma^\sharp) = \left\{ \sigma \mid \forall b \in \mathbb{B},\ \forall i \in [O, size(b)[,\ \sigma(b,i) \in \gamma(\sigma^\sharp(b,i)) \right\}$$

An abstract value is pair of potential memory blocks and an interval

$$Val^\sharp \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{B}) \times \text{Int}$$

For $s \subseteq \mathbb{B}$ and $i \in \text{Int}$,

$$\gamma((s,i)) = \{\mathsf{Num}(n) \mid n \in \gamma(i)\} \cup \{\mathsf{Ptr}(b,n) \mid b \in s,\ n \in \gamma(i)\}$$

An abstract environment is still a map from variable to abstract values

$$Env^\sharp \stackrel{\text{def}}{=} \mathbb{V} \rightarrow Val^\sharp$$

2. For alternative designs, see Sandrine Blazy, Vincent Laporte, David Pichardie. *An Abstract Memory Functor for Verified C Static Analyzers*. ICFP'16

Forward abstract evaluation of expressions (updated)

$$
\begin{array}{rcl}
\mathcal{A}[\![n]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) & = & (\emptyset,[n,n]) \\
\mathcal{A}[\![x]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) & = & \rho^\sharp(x) \\
\mathcal{A}[\![e_1\,o\,e_2]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) & = & \left(s_1, o^\sharp(i_1,i_2)\right) \\
 & & \text{where } (s_1,i_1) = \mathcal{A}[\![e_1]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) \\
 & & \text{and } (s_2,i_2) = \mathcal{A}[\![e_2]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) \\
\mathcal{A}[\![\&X]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) & = & (\{X\},[0,0]) \\
\mathcal{A}[\![*e]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) & = & \left(\bigcup\{s' \mid \exists i',\ (s',i') \in S\}, \bigsqcup\{i' \mid \exists s',\ (s',i') \in S\}\right) \\
 & & \text{where } (s,i) = \mathcal{A}[\![e]\!]^\sharp\,(\rho^\sharp,\sigma^\sharp) \\
 & & \text{and } S = \left\{\sigma^\sharp(b,n) \mid \exists b \in s,\ \exists n \in \gamma(i) \cap [0, size(b)[\right\}
\end{array}
$$

Abstract abstract operator $[\![*e := a]\!]^\sharp$

$$[\![*e := a]\!]^\sharp \, (\rho^\sharp, \sigma_1^\sharp) = (\rho^\sharp, \sigma_2^\sharp)$$

with $\sigma_2^\sharp$ defined by studying 2 cases :

1. if $\mathcal{A}[\![e]\!]^\sharp \, (\rho^\sharp, \sigma_1^\sharp) = (\{b_0\}, [n_0, n_0])$ [strong update]
   - $\sigma_2^\sharp(b_0, n_0) = \mathcal{A}[\![a]\!]^\sharp \, (\rho^\sharp, \sigma_1^\sharp)$
   - $\sigma_2^\sharp(b, n) = \sigma_1^\sharp(b, n) \; \forall (b, n) \neq (b_0, n_0)$
2. otherwise $\mathcal{A}[\![e]\!]^\sharp \, (\rho^\sharp, \sigma_1^\sharp) = (s, i)$ [weak update]
   - $\sigma_2^\sharp(b, n) = \sigma_1^\sharp(b, n) \sqcup \mathcal{A}[\![a]\!]^\sharp \, (\rho^\sharp, \sigma_1^\sharp), \; \forall b \in s, n \in \gamma(i) \cap [0, size(b)[$
   - $\sigma_2^\sharp(b, n) = \sigma_1^\sharp(b, n)$ otherwise.

Outline

Polyhedral abstract interpretation

Automatic discovery of linear restraints among variables of a program.
P. Cousot and N. Halbwachs. POPL'78.

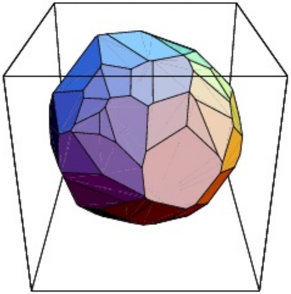
Patrick Cousot

Nicolas Halbwachs

Polyhedral analysis seeks to discover invariant linear equality and inequality relationships among the variables of an imperative program.

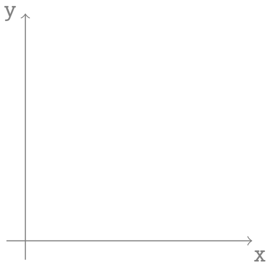
Convex polyhedra

A convex polyhedron can be defined algebraically as the set of solutions of a system of linear inequalities.
Geometrically, it can be defined as a finite intersection of half-spaces.



Polyhedral analysis

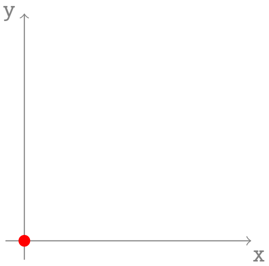
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;


while (x<6) {
  if (?) {

    y = y+2;

  };


  x = x+1;

}
```

Polyhedral analysis

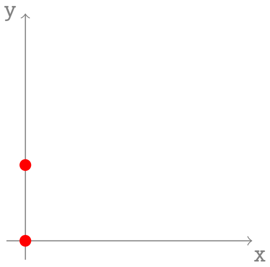
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x = 0 ∧ y = 0}

while (x<6) {
  if (?) {
        {x = 0 ∧ y = 0}
    y = y+2;

  };


  x = x+1;

}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

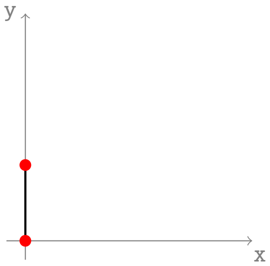


At junction points, we over-approximates union by a convex union.

```
x = 0; y = 0;
      {x = 0 ∧ y = 0}

while (x<6) {
  if (?) {
      {x = 0 ∧ y = 0}
    y = y+2;
      {x = 0 ∧ y = 2}
  };
      {x = 0 ∧ y = 0} ⊎ {x = 0 ∧ y = 2}

  x = x+1;

}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

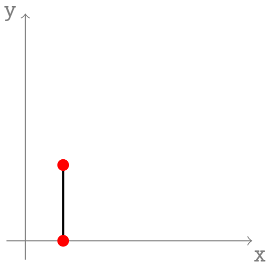


At junction points, we over-approximates union by a convex union.

```
x = 0; y = 0;
          {x = 0 ∧ y = 0}

while (x<6) {
  if (?) {
          {x = 0 ∧ y = 0}
    y = y+2;
          {x = 0 ∧ y = 2}
  };
          {x = 0 ∧ 0 ⩽ y ⩽ 2}

  x = x+1;

}
```

Polyhedral analysis

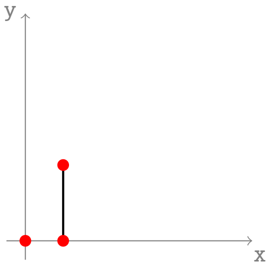
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x = 0 ∧ y = 0}

while (x<6) {
  if (?) {
        {x = 0 ∧ y = 0}
    y = y+2;
        {x = 0 ∧ y = 2}
  };
        {x = 0 ∧ 0 ⩽ y ⩽ 2}

  x = x+1;
        {x = 1 ∧ 0 ⩽ y ⩽ 2}
}
```

Polyhedral analysis

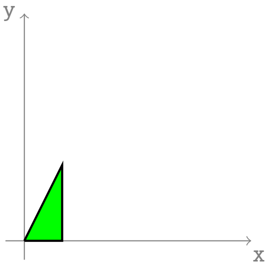
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x = 0 ∧ y = 0} ⊎ {x = 1 ∧ 0 ⩽ y ⩽ 2}

while (x<6) {
  if (?) {
        {x = 0 ∧ y = 0}
    y = y+2;
        {x = 0 ∧ y = 2}
  };
        {x = 0 ∧ 0 ⩽ y ⩽ 2}

  x = x+1;
        {x = 1 ∧ 0 ⩽ y ⩽ 2}
}
```

Polyhedral analysis

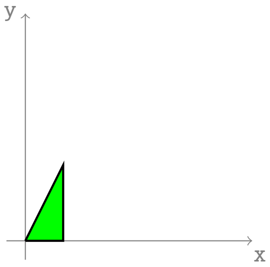
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {$x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x$}

while (x<6) {
  if (?) {
        {$x = 0 \wedge y = 0$}
    y = y+2;
        {$x = 0 \wedge y = 2$}
  };
        {$x = 0 \wedge 0 \leqslant y \leqslant 2$}

  x = x+1;
        {$x = 1 \wedge 0 \leqslant y \leqslant 2$}
}
```

Polyhedral analysis

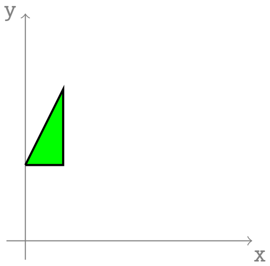
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}

while (x<6) {
  if (?) {
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
    y = y+2;
        {x = 0 ∧ y = 2}
  };
        {x = 0 ∧ 0 ⩽ y ⩽ 2}

  x = x+1;
        {x = 1 ∧ 0 ⩽ y ⩽ 2}
}
```

Polyhedral analysis

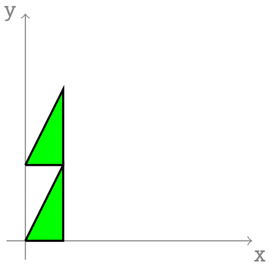
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}

while (x<6) {
  if (?) {
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
    y = y+2;
        {x ⩽ 1 ∧ 2 ⩽ y ⩽ 2x + 2}
  };
        {x = 0 ∧ 0 ⩽ y ⩽ 2}

  x = x+1;
        {x = 1 ∧ 0 ⩽ y ⩽ 2}
}
```

Polyhedral analysis

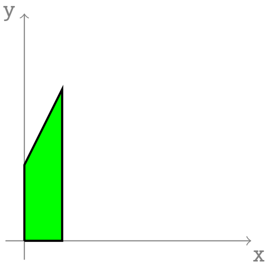
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {$x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x$}

while (x<6) {
  if (?) {
        {$x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x$}
    y = y+2;
        {$x \leqslant 1 \wedge 2 \leqslant y \leqslant 2x + 2$}
  };
        {$x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x$}
                 $\uplus$ {$x \leqslant 1 \wedge 2 \leqslant y \leqslant 2x + 2$}
  x = x+1;
        {$x = 1 \wedge 0 \leqslant y \leqslant 2$}
}
```

Polyhedral analysis

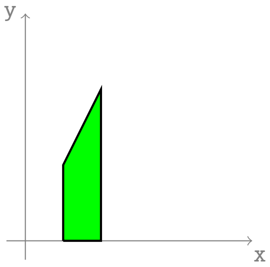
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
          $\{x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x\}$

while (x<6) {
  if (?) {
          $\{x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x\}$
    y = y+2;
          $\{x \leqslant 1 \wedge 2 \leqslant y \leqslant 2x+2\}$
  };
          $\{0 \leqslant x \leqslant 1 \wedge 0 \leqslant y \leqslant 2x+2\}$

  x = x+1;
          $\{x = 1 \wedge 0 \leqslant y \leqslant 2\}$
}
```

Polyhedral analysis

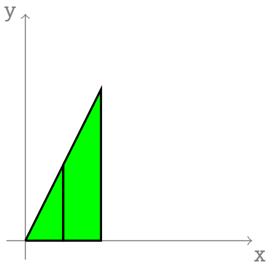
State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



```
x = 0; y = 0;
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}

while (x<6) {
  if (?) {
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
    y = y+2;
        {x ⩽ 1 ∧ 2 ⩽ y ⩽ 2x + 2}
  };
        {0 ⩽ x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x + 2}

  x = x+1;
        {1 ⩽ x ⩽ 2 ∧ 0 ⩽ y ⩽ 2x}
}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

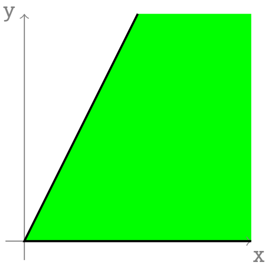


At loop headers, we use heuristics (widening) to ensure finite convergence.

```
x = 0; y = 0;
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
                          ∇ {x ⩽ 2 ∧ 0 ⩽ y ⩽ 2x}
while (x<6) {
  if (?) {
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
    y = y+2;
        {x ⩽ 1 ∧ 2 ⩽ y ⩽ 2x + 2}
  };
        {0 ⩽ x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x + 2}

  x = x+1;
        {1 ⩽ x ⩽ 2 ∧ 0 ⩽ y ⩽ 2x}
}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.



At loop headers, we use heuristics (widening) to ensure finite convergence.

```
x = 0; y = 0;
        {0 ⩽ y ⩽ 2x}

while (x<6) {
  if (?) {
        {x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x}
    y = y+2;
        {x ⩽ 1 ∧ 2 ⩽ y ⩽ 2x + 2}
  };
        {0 ⩽ x ⩽ 1 ∧ 0 ⩽ y ⩽ 2x + 2}

  x = x+1;
        {1 ⩽ x ⩽ 2 ∧ 0 ⩽ y ⩽ 2x}
}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

By propagation we obtain a post-fixpoint

```
x = 0; y = 0;
        {0 ⩽ y ⩽ 2x}

while (x<6) {
  if (?) {
        {0 ⩽ y ⩽ 2x ∧ x ⩽ 5}
    y = y+2;
        {2 ⩽ y ⩽ 2x + 2 ∧ x ⩽ 5}
  };
        {0 ⩽ y ⩽ 2x + 2 ∧ 0 ⩽ x ⩽ 5}

  x = x+1;
        {0 ⩽ y ⩽ 2x ∧ 1 ⩽ x ⩽ 6}
}
        {0 ⩽ y ⩽ 2x ∧ 6 ⩽ x}
```

Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

By propagation we obtain a post-fixpoint which is enhanced by downward iteration.

```
x = 0; y = 0;
        {0 ⩽ y ⩽ 2x ∧ x ⩽ 6}

while (x<6) {
  if (?) {
        {0 ⩽ y ⩽ 2x ∧ x ⩽ 5}
    y = y+2;
        {2 ⩽ y ⩽ 2x + 2 ∧ x ⩽ 5}
  };
        {0 ⩽ y ⩽ 2x + 2 ∧ 0 ⩽ x ⩽ 5}

  x = x+1;
        {0 ⩽ y ⩽ 2x ∧ 1 ⩽ x ⩽ 6}
}
        {0 ⩽ y ⩽ 2x ∧ 6 = x}
```

Polyhedral analysis

A more complex example.

The analysis accepts to replace some constants by parameters.

```
x = 0; y = A;
      $\{A \leqslant y \leqslant 2x + A \wedge x \leqslant N\}$

while (x<N) {
  if (?) {
      $\{A \leqslant y \leqslant 2x + A \wedge x \leqslant N - 1\}$
    y = y+2;
      $\{A + 2 \leqslant y \leqslant 2x + A + 2 \wedge x \leqslant N - 1\}$
  };
      $\{A \leqslant y \leqslant 2x + A + 2 \wedge 0 \leqslant x \leqslant N - 1\}$

  x = x+1;
      $\{A \leqslant y \leqslant 2x + A \wedge 1 \leqslant x \leqslant N\}$
}
      $\{A \leqslant y \leqslant 2x + A \wedge N = x\}$
```

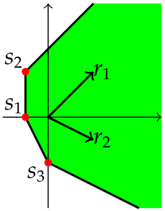
The four polyhedra operations

- $\uplus \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: convex union
  - over-approximates the concrete union at junction points
- $\cap \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: intersection
  - over-approximates the concrete intersection after a conditional intruction
- $[\![\mathtt{x} := e]\!] \in \mathbb{P}_n \to \mathbb{P}_n$: affine transformation
  - over-approximates the assignment of a variable by a linear expression
- $\nabla \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: widening
  - ensures (and accelerates) convergence of (post-)fixpoint iteration
  - includes heuristics to infer loop invariants

```
x = 0; y = 0;
        $P_0 = [\![\mathtt{y} := \mathtt{0}]\!]\,[\![\mathtt{x} := \mathtt{0}]\!]\,(\mathbb{Q}^2) \,\nabla\, P_4$
while (x<6) {
  if (?) {
        $P_1 = P_0 \cap \{x < 6\}$
    y = y+2;
        $P_2 = [\![\mathtt{y} := \mathtt{y} + \mathtt{2}]\!]\,(P_1)$
  };
        $P_3 = P_1 \uplus P_2$
  x = x+1;
        $P_4 = [\![\mathtt{x} := \mathtt{x} + \mathtt{1}]\!]\,(P_3)$
}
        $P_5 = P_0 \cap \{x \geqslant 6\}$
```

Library for manipulating polyhedra

- Parma Polyhedra Library (PPL), NewPolka : complex C/C++ libraries
- They rely on the Double Description Method
  - polyhedra are managed using two representations in parallel



- by set of inequalities

$$P = \left\{ (x,y) \in \mathbb{Q}^2 \;\middle|\; \begin{array}{l} x \geqslant -1 \\ x - y \geqslant -3 \\ 2x + y \geqslant -2 \\ x + 2y \geqslant -4 \end{array} \right\}$$

- by set of generators

$$P = \left\{ \lambda_1 s_1 + \lambda_2 s_2 + \lambda_3 s_3 + \mu_1 r_1 + \mu_2 r_2 \in \mathbb{Q}^2 \;\middle|\; \begin{array}{l} \lambda_1, \lambda_2, \lambda_3, \mu_1, \mu_2 \in \mathbb{R}^+ \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{array} \right\}$$

  - operations efficiency strongly depends on the chosen representations, so they keep both