CTRLVERIF. Analysis of control systems

Lecture 1. Set-based computation and abstract interpretation.

Eric Goubault and Sylvie Putot

MPRI

Outline

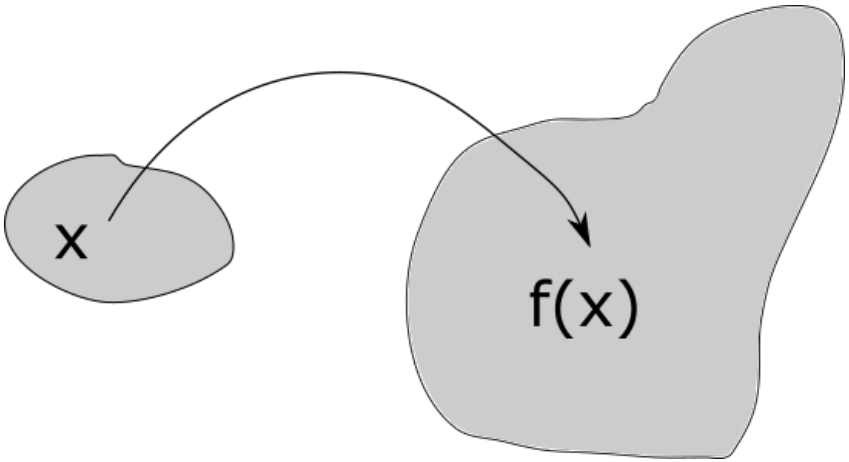
- Set-based methods: interval arithmetic
- Abstract interpretation: numerical abstract domains
- Affine arithmetic and the zonotope abstract domain
- Application to the analysis of numerical properties of programs

Function image computation

Given

- $f : \mathbb{R}^m \to \mathbb{R}^n$
- a set $\mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$

we want:
$f(\mathcal{X}) = \{f(x), x \in \mathcal{X}\}$.

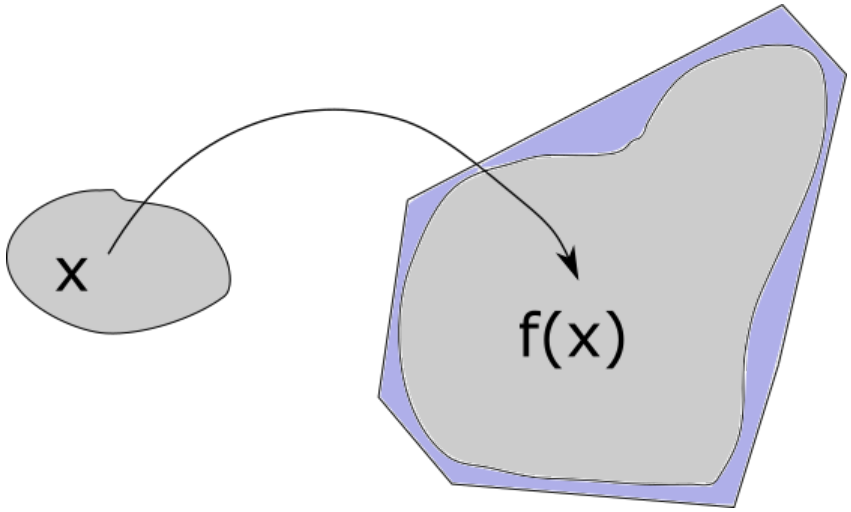


Function image computation

Given

- $f : \mathbb{R}^m \to \mathbb{R}^n$
- a set $\mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$

we want:
$f(\mathcal{X}) = \{f(x), x \in \mathcal{X}\}$.



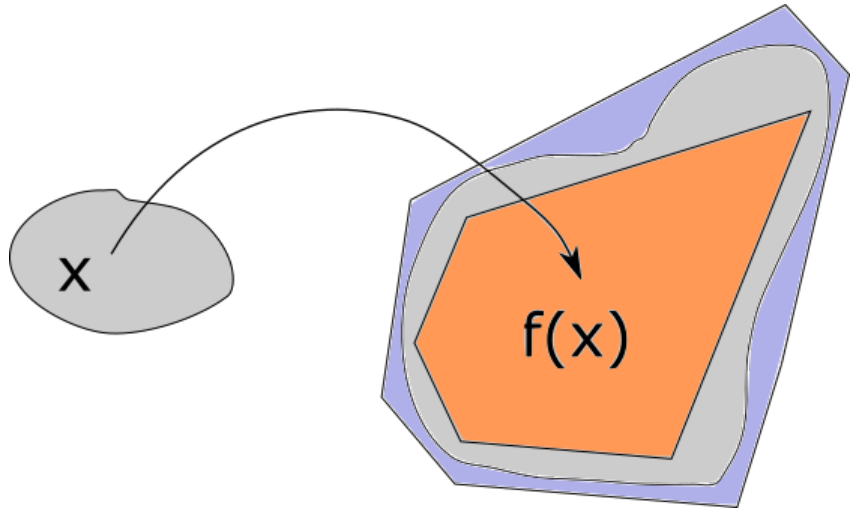
- Over-approximating extension of f (or inclusion function):
  $\boldsymbol{f_o} : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$ such that $\forall \mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m), f(\mathcal{X}) \subseteq \boldsymbol{f_o}(\mathcal{X})$
  = states that may belong to the image by the function of some input set

Function image computation

Given

- $f : \mathbb{R}^m \to \mathbb{R}^n$
- a set $\mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$

we want:
$f(\mathcal{X}) = \{f(x), x \in \mathcal{X}\}$.



- Over-approximating extension of f (or inclusion function):
  $\boldsymbol{f}_o : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$ such that $\forall \mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$, $f(\mathcal{X}) \subseteq \boldsymbol{f}_o(\mathcal{X})$
  = states that may belong to the image by the function of some input set
- Under-approximating extension of f:
  $\boldsymbol{f}_u : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$ such that $\forall \mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$, $\boldsymbol{f}_u(\mathcal{X}) \subseteq f(\mathcal{X})$
  =states proved to belong to the image by the function of some input set

Function image computation

- Over-approximating extension of f (or inclusion function):
  $\boldsymbol{f_o} : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$ such that $\forall \mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$, $f(\mathcal{X}) \subseteq \boldsymbol{f_o}(\mathcal{X})$
  = *states that may belong to the image by the function of some input set*
- Under-approximating extension of f:
  $\boldsymbol{f_u} : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$ such that $\forall \mathcal{X}$ in $\mathcal{P}(\mathbb{R}^m)$, $\boldsymbol{f_u}(\mathcal{X}) \subseteq f(\mathcal{X})$
  =*states proved to belong to the image by the function of some input set*

First focus on the much more classical over-approximations

Computing inclusion functions: interval arithmetic

Interval arithmetic: replace each number by an interval containing it and compute

- So that the exact result is guaranteed to be contained in the computed interval
- Initially (Moore 1966): small intervals, introduced to take into account roundoff errors as well as uncertainties (on the physical data. . .)
- Later: more generally, computations with large sets (with specific algorithms)

Interval analysis: develop dedicated algorithms for reliable (or verified/guaranteed/certified) computing

- Control or quantify roundoff errors
- Compute certified solutions of problems such as constraint satisfaction, global optimization, ODE integration
- Mathematical proofs (e.g. Hale's proof of Kepler's conjecture)

Intervals

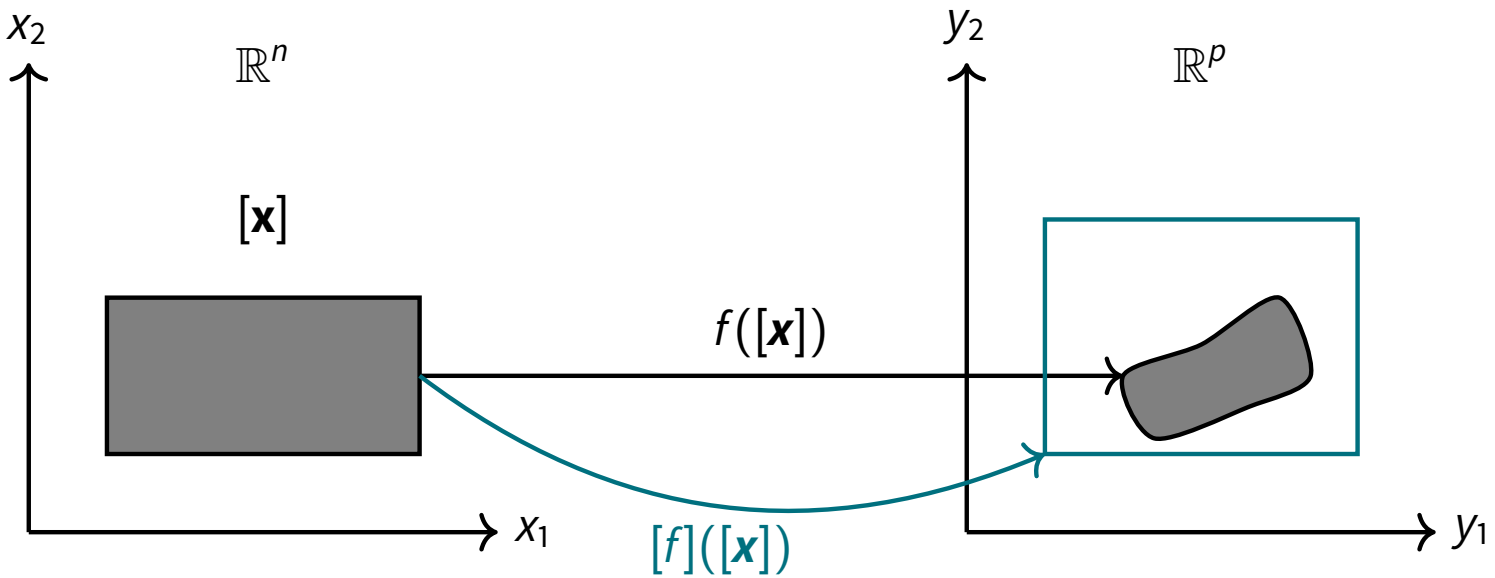
- Interval of real numbers $[x] \in \mathbb{IR} : [x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$
  - interval for π on a radix-10 machine with 3 digits of mantissa : $[3.14159, 3.14160]$
  - data d measured with absolute error ε : $[d - \varepsilon, d + \varepsilon]$
  - more generally, for $S \in \mathcal{P}(\mathbb{R})$: $[\inf S, \sup S]$
- Vector of intervals = Box = Cartesian product $[\boldsymbol{x}] \in \mathbb{IR}^n : [\boldsymbol{x}] = ([x_1], \ldots, [x_n])^t$

Inclusion function

Definition

For $f : \mathbb{R}^n \to \mathbb{R}^p$, an interval extension or inclusion function is an interval function $[f] : \mathbb{IR}^n \to \mathbb{IR}^p$ such that

$$\forall [\boldsymbol{x}] \in \mathbb{IR}^n, f([\boldsymbol{x}]) = \{f(\boldsymbol{x}), \boldsymbol{x} \in [\boldsymbol{x}]\} \subseteq [f]([\boldsymbol{x}])$$

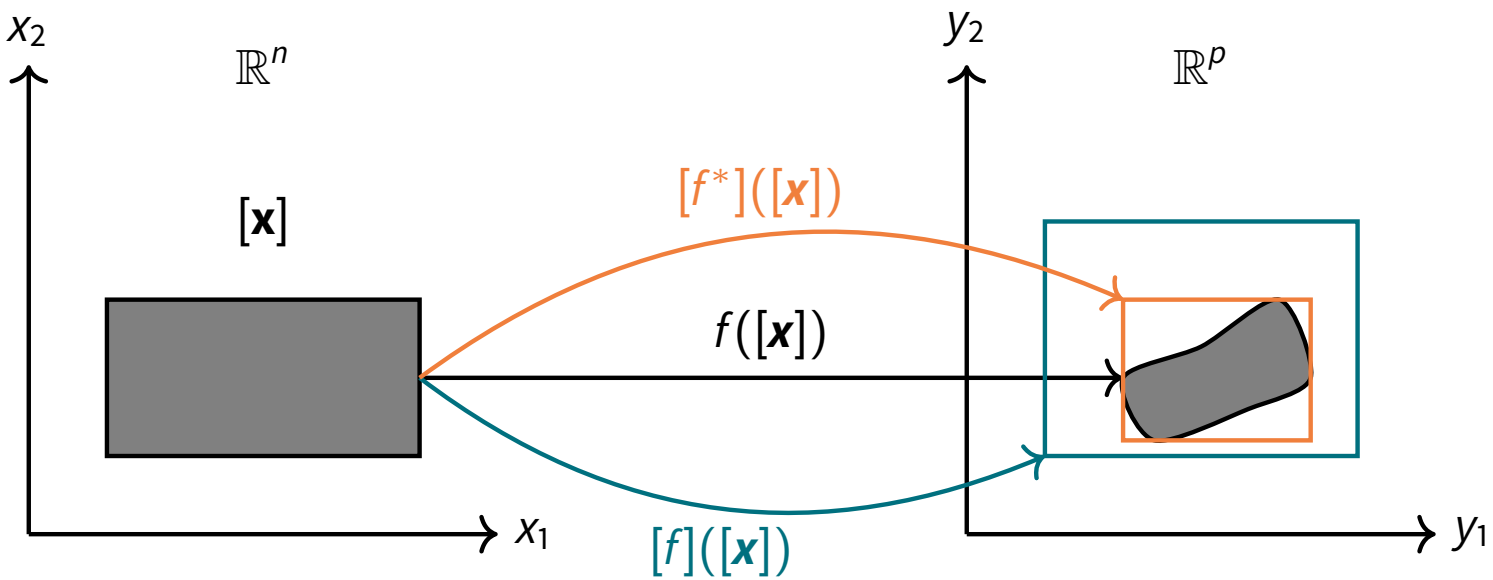


Inclusion function

Definition

For $f : \mathbb{R}^n \to \mathbb{R}^p$, an interval extension or inclusion function is an interval function $[f] : \mathbb{IR}^n \to \mathbb{IR}^p$ such that

$$\forall [\boldsymbol{x}] \in \mathbb{IR}^n, f([\boldsymbol{x}]) = \{f(\boldsymbol{x}), \boldsymbol{x} \in [\boldsymbol{x}]\} \subseteq [f]([\boldsymbol{x}])$$



We denote $[f^*]$ the minimal inclusion function associated with f, such that $[f^*]([\boldsymbol{x}])$ is the smallest box enclosing $f([\boldsymbol{x}])$ for all $[\boldsymbol{x}]$.

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

- $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] =?$

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

- $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = [\min(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y}), \max(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y})]$

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

- $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = [\min(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y}), \max(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y})]$
- $[\underline{x}, \overline{x}]^2 = [\min(\underline{x}^2, \overline{x}^2), \max(\underline{x}^2, \overline{x}^2)]$ if $0 \notin [\underline{x}, \overline{x}]$,
  $[0, \max(\underline{x}^2, \overline{x}^2)]$ otherwise

The natural interval extension or interval arithmetic

Interval arithmetic: the arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval

$$\text{for } \circ = +, -, \times, / : \ \{x \circ y \mid x \in [x], y \in [y]\} \subseteq [x] \circ [y]$$

Using monotonicity, interval extensions with the interval endpoints:

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

Beware ! $[x] - [x] = \{x_1 - x_2 \mid x_1 \in [x], x_2 \in [x]\} \neq 0$.

- $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = [\min(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y}), \max(\underline{x} \times \underline{y}, \underline{x} \times \overline{y}, \overline{x} \times \underline{y}, \overline{x} \times \overline{y})]$
- $[\underline{x}, \overline{x}]^2 = [\min(\underline{x}^2, \overline{x}^2), \max(\underline{x}^2, \overline{x}^2)]$ if $0 \notin [\underline{x}, \overline{x}]$,
  $[0, \max(\underline{x}^2, \overline{x}^2)]$ otherwise
- $[\underline{x}, \overline{x}]/[\underline{y}, \overline{y}] = [\underline{x}, \overline{x}] \times [\min(1/\underline{y}, 1/\overline{y}), \max(1/\underline{y}, 1/\overline{y})]$ if $0 \notin [\underline{y}, \overline{y}]$

Algebraic properties

Preserved:

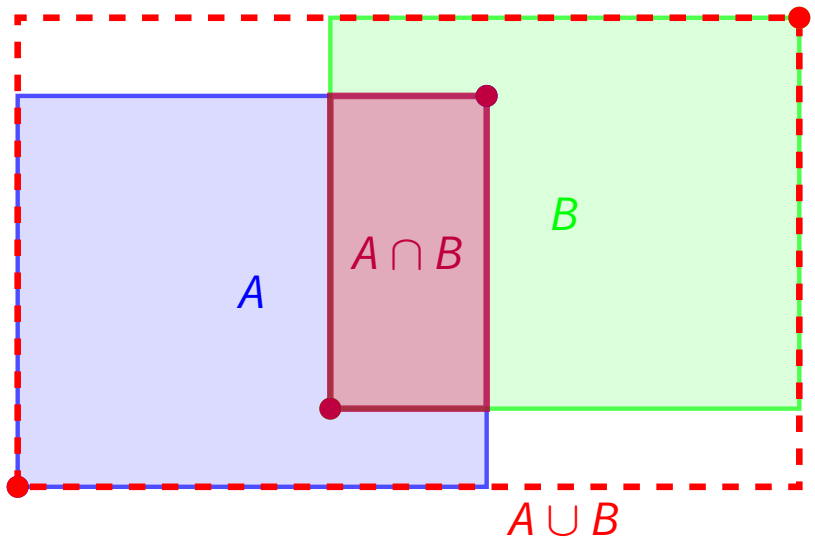
- associativity, commutativity hold

Lost:

- subtraction $\neq$ addition^{-1}
  - in particular $[x] - [x] \supsetneq [0, 0]$ in general (when $\underline{x} \neq \overline{x}$)
- division $\neq$ multiplication^{-1}
- squaring is tighter in general (when $0 \in [x]$) than multiplication by oneself
  - $[-1, 1] \times [-1, 1] = [-1, 1]$ while $[-1, 1]^2 = [0, 1]$
- multiplication is only sub-distributive wrt addition
  - $[x] \times ([y] + [z]) \subset [x] \times [y] + [x] \times [z]$

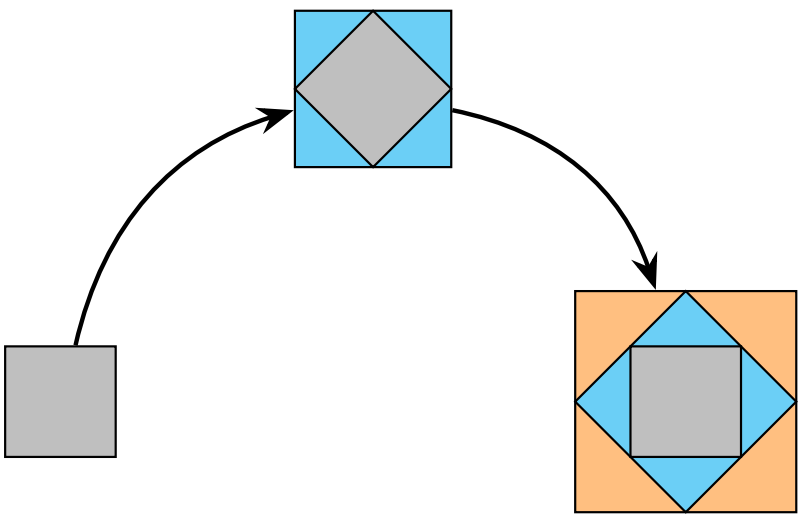
Other interval operators

- elementary functions (exp, log, sin, . . .): again use (piecewise) monotony
- set operators: union, intersection



Interval arithmetic: sources of overestimation

- Interval arithmetic does not define a minimum inclusion function (which would compute the smallest box that contains $\{f(\boldsymbol{x}), \boldsymbol{x} \in [\boldsymbol{x}]\}$)
- The main source of overestimation is decorrelation of variables, known as dependency problem. Typically, $[x] - [x] = \{x - y \mid x \in [x], y \in [y]\} \neq 0$
- The geometric view, known as wrapping effect: boxes are non relational, rectangles with sides parallel to the axes cannot capture relations between dimensions



- gray: initial box and its exact image through 2 successive rotations
- cyan: in the center, the best box abstraction of the exact gray image; on the right, its image by the 2nd rotation
- orange: the best box abstraction of the blue image (which is NOT the minimal box enclosing the exact image in gray)

General function inclusion by interval arithmetic

- For a general (non monotonic) function, cannot reason on the interval endpoints
- Inclusion by interval arithmetic, also known as natural interval extension: replace operators $+, -, \times, /$ and elementary functions by interval counterparts
- Problem: different results for mathematically equivalent expressions (in real nb)
  - Always correct, but the enclosure can be unsatisfyingly wide
  - How to choose the best extension ? How to choose a good one ?

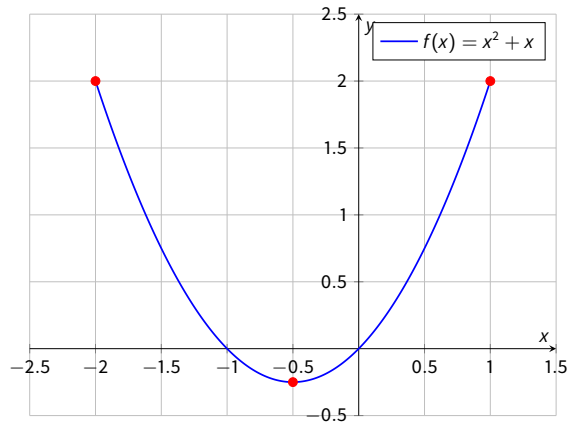
Example: evaluate $f(x) = x(x+1)$ for $[x] = [-2, 1]$. Can you rewrite the expression to improve the inclusion ?

General function inclusion by interval arithmetic

- For a general (non monotonic) function, cannot reason on the interval endpoints
- Inclusion by interval arithmetic, also known as natural interval extension: replace operators $+, -, \times, /$ and elementary functions by interval counterparts
- Problem: different results for mathematically equivalent expressions (in real nb)
  - Always correct, but the enclosure can be unsatisfyingly wide
  - How to choose the best extension ? How to choose a good one ?

Example: evaluate $f(x) = x(x+1)$ for $[x] = [-2, 1]$. Can you rewrite the expression to improve the inclusion ?

- evaluate $f(x) = x(x+1)$ for $[x] = [-2, 1]$:
  $[f]([x]) = [-2, 1]([-2, 1]+1) = [-2, 1] \times [-1, 2] = [-4, 2]$
- but writing $f(x) = x^2 + x$:
  $[-2, 1]^2 + [-2, 1] = [0, 4] + [-2, 1] = [-2, 5]$
- and writing $f(x) = (x + \frac{1}{2})^2 - \frac{1}{4}$:
  $([-2, 1] + \frac{1}{2})^2 - \frac{1}{4} = [-\frac{3}{2}, \frac{3}{2}]^2 - \frac{1}{4} = [0, \frac{9}{4}] - \frac{1}{4} = [-\frac{1}{4}, 2]$
  which is the tightest result: $f(-\frac{1}{2}) = -\frac{1}{4}$ and $f(1) = 2$



Approximation properties of the natural interval extension

We define the Hausdorff distance on intervals by

$$\begin{aligned} q([x],[y]) &= \min\{q \in \mathbb{R}^+ \mid [x] \subset [y] + [-q,q] \wedge [y] \subset [x] + [-q,q]\} \\ &= \max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|) \end{aligned}$$

and $w([x]) = \overline{x} - \underline{x}$ the width of $[x]$.

Theorem (Moore 1966)

For f a function of n variables defined by an arithmetic expression which is Lipschitz in th interval sense for $[\boldsymbol{x}_0] \in \boldsymbol{I}^n$, then we have, for all $[\boldsymbol{x}] \subset [\boldsymbol{x}_0]$:

$$q([f]_{Nat}([x]), f([\boldsymbol{x}])) = \mathcal{O}(w([x]))$$

and

$$w(f([\boldsymbol{x}]) \leq \lambda_f([\boldsymbol{x}_0])w([\boldsymbol{x}])$$

where $\lambda_f([\boldsymbol{x}_0])$ is the Lipschitz constant of f on $[\boldsymbol{x}_0]$, which depends only on f and $[\boldsymbol{x}_0]$.
R. Moore. Interval analysis. Prentice Hall, 1966

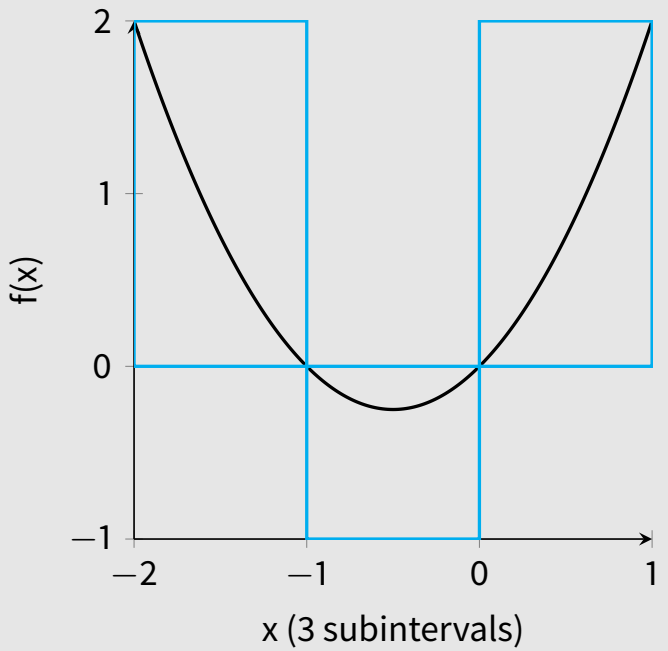
Thus the natural extension is a linear approximation: the width of a function enclosure is proportional to the width of the input interval

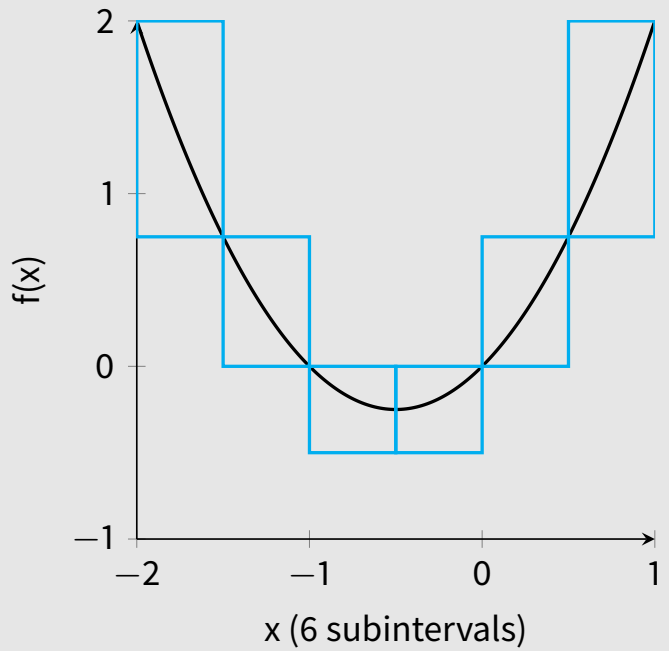
First solution: splitting the input boxes

- Split the input intervals into small subintervals, evaluate the function on each subinterval and take the union of these partial results, thus paving the resulting set.

Example

Evaluate $f(x) = x(x+1)$ for $[x] = [-2, 1]$ (remember that $[f]([-2, 1]) = [-4, 2]$)





Gaganov 1982: evaluation up to ϵ of a multivariate polynomial with rational coeff. on a box is NP-hard

Second solution: Mean-value interval extension

Theorem (Mean-value theorem (1st order Taylor-Lagrange expansion))

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function on $[a, b]$, differentiable on (a, b), where $a < b$. Then there exists $c \in (a, b)$ such that

$$f(b) = f(a) + (b - a)f'(c)$$

Can you define a mean-value extension ? Apply it to $f(x) = x^2 - x$ for $x \in [2, 3]$

Second solution: Mean-value interval extension

Theorem (Mean-value theorem (1st order Taylor-Lagrange expansion))

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function on $[a, b]$, differentiable on (a, b), where $a < b$. Then there exists $c \in (a, b)$ such that

$$f(b) = f(a) + (b - a)f'(c)$$

Can you define a mean-value extension ? Apply it to $f(x) = x^2 - x$ for $x \in [2, 3]$ Interval version: mean-value interval extension

$$\forall x \in [x], f([\boldsymbol{x}]) \subseteq f(x) + ([x] - x).[f']([x])$$

Centered form (Moore): when x is taken as the interval midpoint $m([x]) = \frac{\underline{x}+\overline{x}}{2}$

$$[f]_{TL1}([x]) = f(m([x])) + ([x] - m([x])).[f']([x])$$

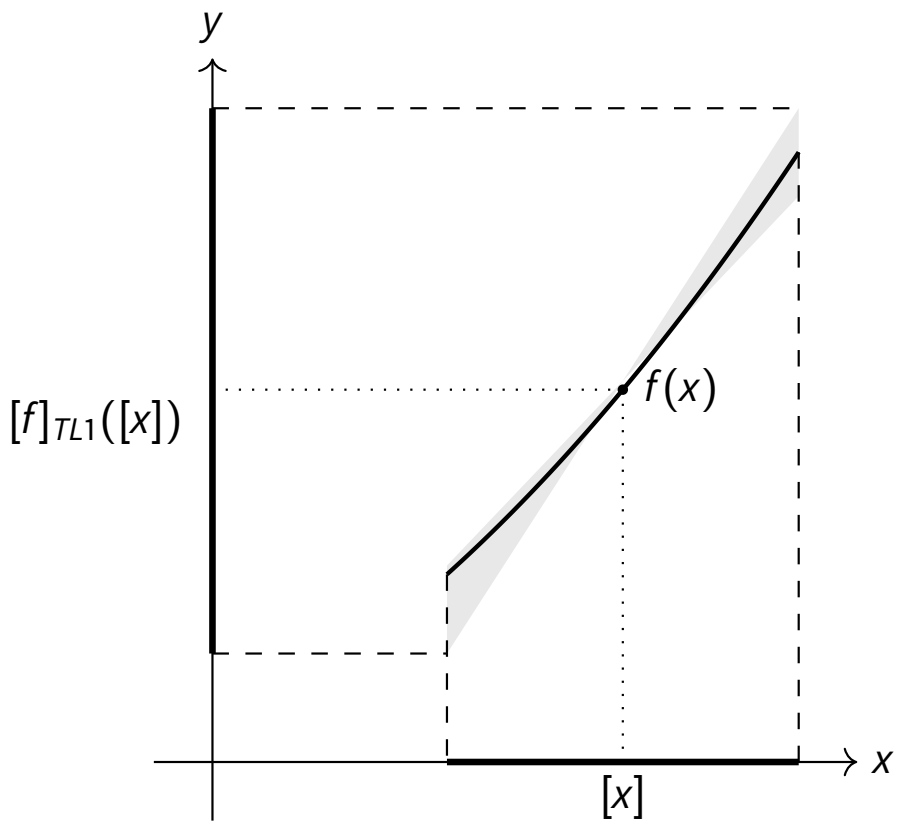
Naturally extends to $f : \mathbb{R}^m \to \mathbb{R}$:

$$[f]_{TL1}([x]) = f(m([x])) + \sum_{i=1}^{m}([x_i] - m([x_i])).[f']([x]) = f(m([x])) + ([x] - m([x])).[J_f]([x])$$

Mean-value extension: illustration

For $f(x) = x^2 - x$ for $x \in [2,3]$: we have $f(2.5) = 3.75$ and $|f'([2,3])| \subseteq [3,5]$. Then

$f([2,3]) \subseteq 3.75 + ([2,3] - 2.5) * [2,3] = 3.75 + [-0.5, 0.5] * [3,5] = 3.75 + [-2.5, 2.5] = [1.25, 6.25]$



Any idea to also define an inner-approximation using the mean-value theorem ?

Interval mean-value generalized for inner-approximation

Theorem (Generalized Interval Mean-Value Theorem,Goldsztejn 2012)

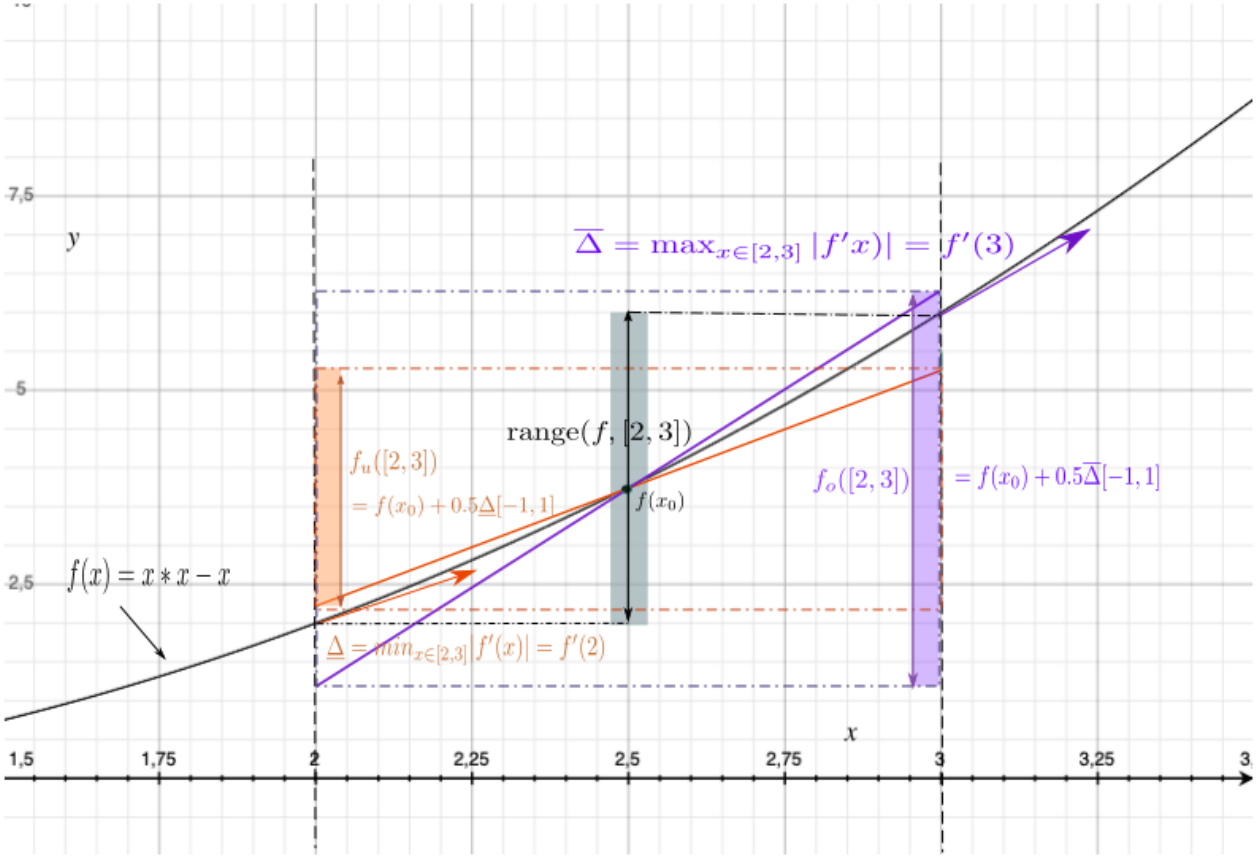
- $f : \mathbb{R}^m \to \mathbb{R}$ *be a continuously differentiable function,* $\boldsymbol{x}$ *an initial box of* $\mathbb{R}^m$*,*
- $x_0 = mid(\boldsymbol{x})$ *the center of the box* x*,* $\boldsymbol{f}_0 = [\underline{f_0}, \overline{f_0}]$ *such that* $f(x_0) \in \boldsymbol{f}_0$
- $\Delta_i = [\underline{\Delta_i}, \overline{\Delta_i}]$ *such that* $\{|f_i'(x_{0,1}, \ldots, x_{0,i-1}, x_i, \ldots, x_m)|,\ x \in \boldsymbol{x}\} \subseteq \Delta_i$

Then:

$$f(\boldsymbol{x}) \subseteq [\underline{f_0}, \overline{f_0}] + \sum_{i=1}^{m} \overline{\Delta_i} radius(\boldsymbol{x}_i)[-1, 1]$$

$$[\overline{f_0} - \sum_{i=1}^{m} \underline{\Delta_i} radius(\boldsymbol{x}_i), \underline{f_0} + \sum_{i=1}^{m} \underline{\Delta_i} radius(\boldsymbol{x}_i)] \subseteq f(\boldsymbol{x})$$

Example: $f(x) = x^2 - x$ over $\boldsymbol{x} = [2, 3]$



$f(2.5) = 3.75$ and $|\boldsymbol{f}'([2, 3])| \subseteq [3, 5] = [\underline{\Delta}, \overline{\Delta}]$. Then,

$$3.75 + 0.5 * 3 * [-1, 1] \subseteq f([2, 3]) \subseteq 3.75 + 0.5 * 5 * [-1, 1]$$

from which we deduce

$$[2.25, 5.25] \subseteq f([2, 3]) \subseteq [1.25, 6.25]$$

Sound implementation

What if I cannot compute exactly, for instance I only know $f(2.5) \in [3.5, 4]$?

Sound implementation

What if I cannot compute exactly, for instance I only know $f(2.5) \in [3.5, 4]$?

$$f([2, 3]) \subseteq [3.5, 4] + 0.5 * 5 * [-1, 1] = [1, 6.5]$$

instead of the tighter $[1.25, 6.25]$.

Sound implementation

What if I cannot compute exactly, for instance I only know $f(2.5) \in [3.5, 4]$?

$$f([2,3]) \subseteq [3.5,4] + 0.5 * 5 * [-1,1] = [1, 6.5]$$

instead of the tighter $[1.25, 6.25]$.

$$[4 - 0.5 * 3, 3.5 + 0.5 * 3] = [2.5, 5] \subseteq f([2,3])$$

instead of the wider $[2.25, 5.25]$.

Sound implementation

What if I cannot compute exactly, for instance I only know $f(2.5) \in [3.5, 4]$?

$$f([2,3]) \subseteq [3.5,4] + 0.5 * 5 * [-1,1] = [1, 6.5]$$

instead of the tighter $[1.25, 6.25]$.

$$[4 - 0.5 * 3, 3.5 + 0.5 * 3] = [2.5, 5] \subseteq f([2,3])$$

instead of the wider $[2.25, 5.25]$.

Wider approximation of f^0 (or the gradient $f'(x)$) leads to:

- wider over-approximation (hence lower quality)
- smaller inner-approximation (hence lower quality)

Means it can be soundly implemented in the general case.

Properties of the centered form

Choice of $x \in [x]$ as the midpoint:

- other choices could be maximize the lower bound or minimize the upper bound of the enclosure, midpoint minimizes the width of the enclosure (Beaumann, 1988)
- if the inclusion of the derivative is monotonic, the midpoint choice yields a monotonic (with respect to inclusion) inclusion function, meaning

$$[x] \subset [y] \implies [f]([x]) \subset [f]([y]),$$

  which is not the case for other choices of x (Caprani and Madsen 1980)

Beaumann. Optimal centered forms. BIT, 1988
Caprani and Madsen. Mean value forms in interval analysis. Computing, 1980

Approximation properties of the centered form interval extension

Theorem (Neumaier 1990)

The inclusion function $[f]_{TL1}$ *satisfies*

$$0 \leq w([f]_{TL1}([x])) - w(f([\boldsymbol{x}])) \leq q([f]_{TL1}([x]), f([\boldsymbol{x}])) \leq \frac{1}{2} w([\boldsymbol{f}'])([x]).w([x])$$

A. Neumaier. Interval methods for systems of equations. Cambridge University Press, 1990

- if $[\boldsymbol{f}'])([x])$ is Lipschitz for intervals, then the order 1 Taylor Lagrange inclusion function is a quadratic approximation.
- while the natural extension is a linear approximation

$\Rightarrow$ not strictly comparable but $TL1$ tends to be better for small intervals and the natural extension for larger intervals

Order 2 extension

Order 2 Taylor-Lagrange expansion:

Let $f : \mathbb{R} \to \mathbb{R}$ be a C^1 function on $[a, b]$, twice differentiable on (a, b), where $a < b$.
Then there exists $c \in (a, b)$ such that

$$f(b) = f(a) + (b - a)f'(a) + \frac{1}{2}(b - a)^2 f''(c)$$

Interval version: order 2 interval extension

$$\forall x \in [x], f([\boldsymbol{x}]) \subseteq f(x) + ([x] - x).f'(x) + \frac{1}{2}([x] - x)^2.[f'']([x])$$

Abstract Interpretation for automatic program verification

- Abstract interpretation is a theory of semantics approximation (P. & R. Cousot, 1977)
- Provides a systematic way to build automated program analyzers
- By synthesizing local abstract invariants fully automatically, from the source code
- Without executing the program, for sets of inputs

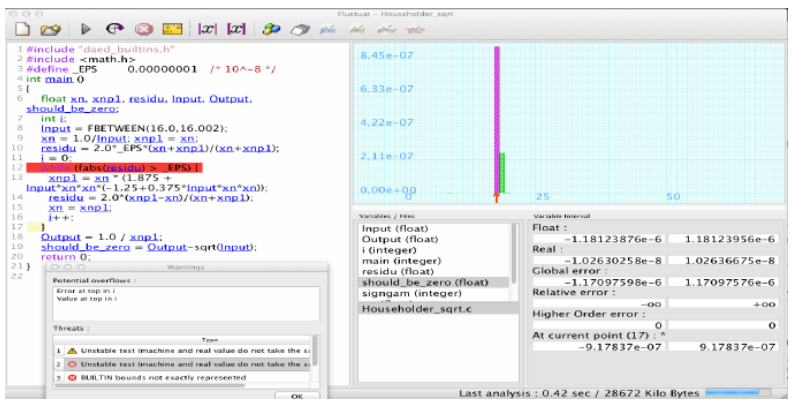
Incomplete but scalable methods

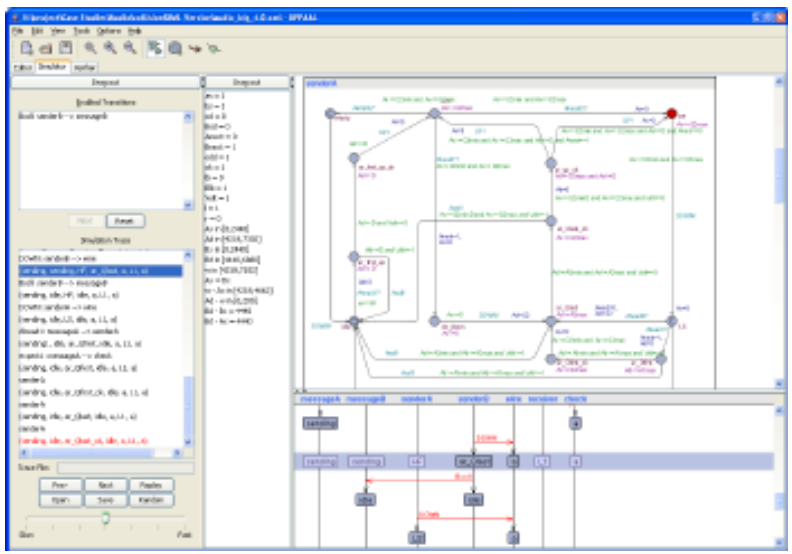
- Abstract program semantics in a conservative way with scalable computation
- A branch of abstractions includes the numerical abstract domains:
  - Abstraction of subsets of R^n and of functions from R^m to R^n
  - Compute sound enveloppes of state variables values
  - Classical abstraction: boxes or intervals, with transfer functions = interval arithmetic

See course "Abstract interpretation: application to verification and static analysis" (Antoine Miné)

Program verification: classes of formal methods

- Abstract interpreters: synthetize properties (in a limited set of "abstract" properties) automatically, directly on programs/"code" of systems
- Model-checkers: checking properties automatically on models of computation/systems; SAT/SMT solvers
- Interactive provers: interactively prove/discover properties on code/system





Automatic program verification?

Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x :=Input[-1,1];
y :=Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$

- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?

Automatic program verification?

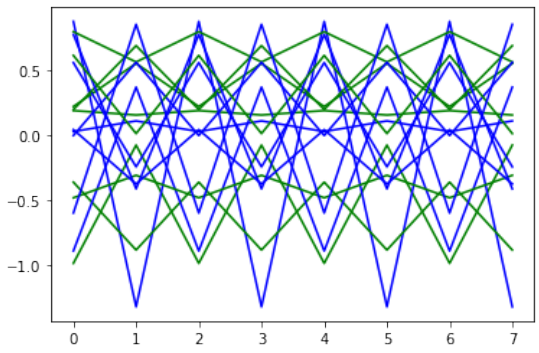
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)

Automatic program verification?

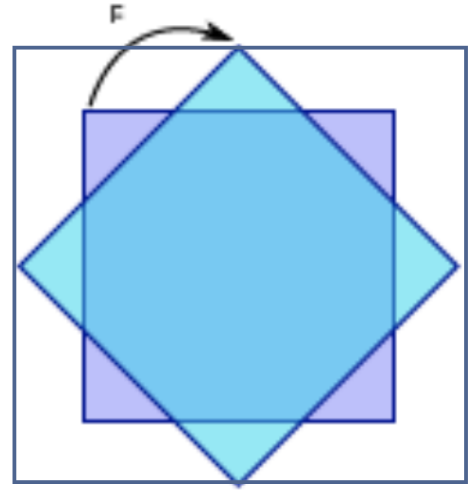
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?
  - Easy if G is an inductive invariant (G holds initially and $F(G) \subseteq G$), but not the case here

Automatic program verification?

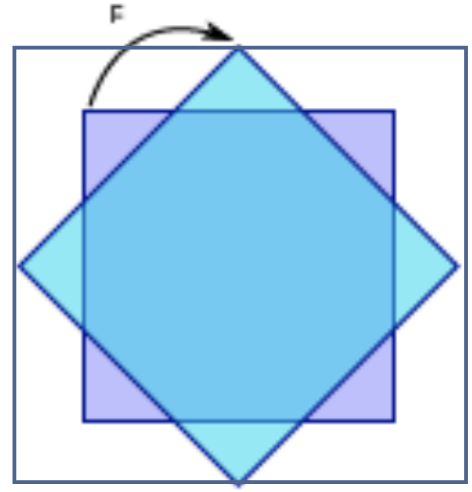
Example: can we prove that $x, y \in [-2, 2]$ is a loop invariant of the loop below?

Program to analyze:

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

Program semantics:

- initial values of (x, y): initial set $I = [-1, 1] \times [-1, 1]$
- loop effect on (x, y): $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ with $F(X) = \{(\sqrt{2}/2 * (x + y), \sqrt{2}/2 * (x - y)) \mid (x, y) \in X\}$



- Is $(x, y) \in G = [-2, 2] \times [-2, 2]$ a loop invariant ?
  - YES! (all executions satisfy $(x, y) \in G$ at any iteration at loop head)
- But how do I prove this ?
  - Easy if G is an inductive invariant (G holds initially and $F(G) \subseteq G$), but not the case here
- And what if I have no idea what an invariant is ?

$\Rightarrow$ Synthesize automatically inductive invariants

Invariant synthesis

We want local numerical invariants = at each program point, properties/values of variables true for all execution traces, and if possible the strongest properties

Example

```
void main() {
  int x := {-100,-99,...,50};
  // X = {-100,-99,...,50}
  while (x<100) {
    // X = {-100,-99,...,99}
    x=x+1;
    // X = {-99,-98,...,100}
  }
   // X = {100}
}
```

Invariant synthesis:

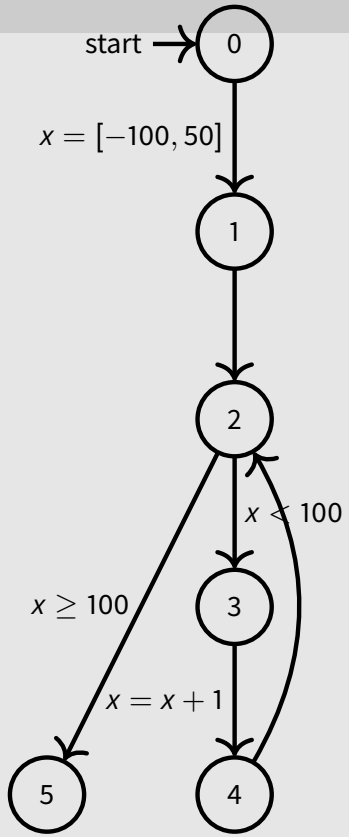
- Program semantics as a discrete dynamical system over the variables values, $F(X)$
- Local invariants = reachable states as solution of fixed point equation $X = F(X)$
- In later courses, we will extend similar techniques for continuous-time systems

Forward collecting semantics - informally

- We construct the control flow graph of a program, as a transition system $\mathcal{T} = (S, i, E, Tran)$
  - S is the set of control points of the program (we define [i] as the control point corresponding to the execution point between line i and line i + 1)
  - $E = \{x = expr \mid Aexp\} \cup Bexp$
  - We define a certain number of rules for the transitions

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1; // [4]
 }          // [5]
}
```



Forward collecting semantics - informally

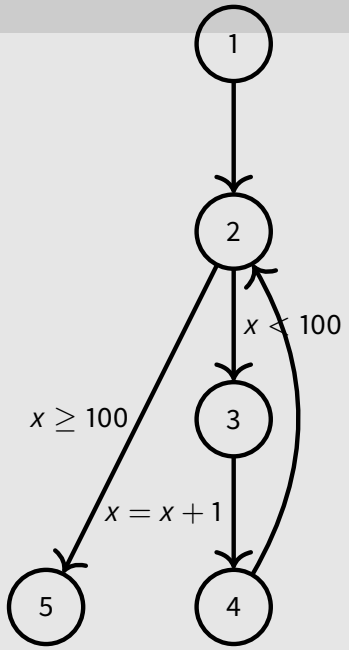
- We associate to each control point s_i of $\mathcal{T}$ an equation in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

where $[\![t]\!]$ is the interpretation of the transition t (/transfer function), seen as a function from the set of values of variables to itself

Example

```
void main()
{ // [0]
  int x=[-100,50];//[1]
  while /*[2]*/(x<100)
  {          // [3]
    x=x+1;   // [4]
  }          // [5]
}
```



$$\begin{aligned} x_0 &= \top \\ x_1 &= \{-100, \ldots, 50\} \\ x_2 &= x_1 \cup x_4 \\ x_3 &=]-\infty, 99] \cap x_2 \\ x_4 &= x_3 + 1 \\ x_5 &= [100, +\infty[\cap x_2 \end{aligned}$$

Forward collecting semantics - informally

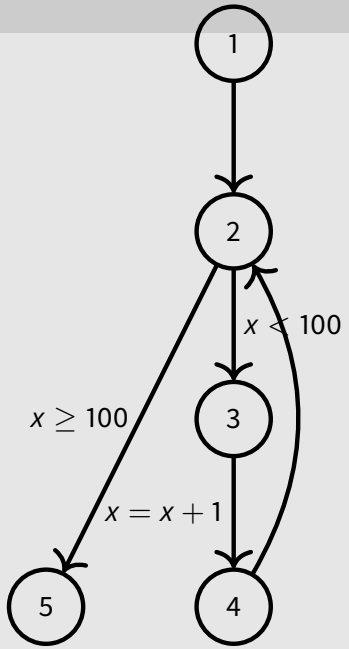
- We associate to each control point s_i of $\mathcal{T}$ an equation in associated variables X_i, which represent the sets of values that program variables can take, at control points s_i, and collect values coming from all paths:

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j$$

- The solutions of this system $X = F(X)$ are the local invariants at control points s_i
  - interested in the strongest properties = the smallest fixpoint

Example

```
void main()
{ // [0]
 int x=[-100,50];//[1]
 while /*[2]*/(x<100)
 {          // [3]
   x=x+1; // [4]
 }          // [5]
}
```



$$\begin{aligned} x_0 &= \top \\ x_1 &= \{-100, \ldots, 50\} \\ x_2 &= x_1 \cup x_4 \\ x_3 &=]-\infty, 99] \cap x_2 \\ x_4 &= x_3 + 1 \\ x_5 &= [100, +\infty[\cap x_2 \end{aligned}$$

Transfer functions for arithmetic expressions

- At each control point $s \in S$, we compute a set of environments $\Sigma \in \wp(Loc \to \boldsymbol{Z})$ reachable for some execution path
- We define the set of values an arithmetic expression can take:
  - $[\![n]\!]\Sigma = \{n\}$
  - $[\![X]\!]\Sigma = \Sigma(X)$ where we write $\Sigma(X) = \{\sigma(X) \mid \sigma \in \Sigma\}$
  - $[\![a_0 + a_1]\!]\Sigma = \{x + y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma,\ \sigma \in \Sigma\}$
  - $[\![a_0 - a_1]\!]\Sigma = \{x - y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma,\ \sigma \in \Sigma\}$
  - $[\![a_0 * a_1]\!]\Sigma = \{x * y \mid x = [\![a_0]\!]\sigma,\ y = [\![a_1]\!]\sigma, \sigma \in \Sigma\}$
- Then: $[\![X := a]\!]\Sigma = \Sigma[[\![a]\!]\Sigma/X]$ where $\Sigma[[\![a]\!]\Sigma/X] = \{\sigma[[\![a]\!]\sigma/X] \mid \sigma \in \Sigma\}$
- Conditionals
  - $[\![true]\!]\Sigma = \Sigma$
  - $[\![false]\!]\Sigma = \bot$ (where $\bot$ represents value "bottom" for the environments), i.e. $\bot(x) = \bot$ for all x
  - $[\![a_0 = a_1]\!]\Sigma = \Sigma_2$ with $\Sigma_2 \subseteq \Sigma$ is the set of environments $\sigma \in \Sigma$ such that $[\![a_0 = a_1]\!]\sigma = true$
  - Similarly for $a_0 < a_1$ etc.

Solving the fixpoint equations?

Existence of the smallest solution of the fixpoint equations $X = F(X)$ defined by

$$X_i = \bigcup_{s_j \in S | (s_j, t, s_i) \in Tran} [\![t]\!] X_j ?$$

Solving the fixpoint equations?

Existence of the smallest solution of the fixpoint equations $X = F(X)$ defined by

$$X_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} [\![t]\!] X_j ?$$

Tarski's theorem

Suppose

- $D = (\wp(Loc \to \boldsymbol{Z}), \subseteq, \cup, \cap, \emptyset, (Loc \to \boldsymbol{Z}))$ is a complete lattice
- F is monotonic in this lattice $(\forall d, d' \in D,\ d \subseteq d' \Rightarrow F(d) \subseteq F(d'))$

then the least fixpoint exists and is unique

Solving the fixpoint equations?

Existence of the smallest solution of the fixpoint equations $X = F(X)$ defined by

$$X_i = \bigcup_{s_j \in S \mid (s_j, t, s_i) \in Tran} [\![t]\!] X_j ?$$

Tarski's theorem

Suppose

- $D = (\wp(Loc \to \boldsymbol{Z}), \subseteq, \cup, \cap, \emptyset, (Loc \to \boldsymbol{Z}))$ is a complete lattice
- F is monotonic in this lattice $(\forall d, d' \in D,\ d \subseteq d' \Rightarrow F(d) \subseteq F(d'))$

then the least fixpoint exists and is unique

Kleene's iteration

If F is continuous on a CPO (F monotonic and $\forall d_0 \subseteq d_1 \subseteq \ldots \subseteq d_n \subseteq \ldots$ of D: $\bigcup_{n \in \boldsymbol{N}} F(d_n) = F\left(\bigcup_{n \in \boldsymbol{N}} d_n\right)$), then the least fixpoint of F can be computed by the increasing iterations:

$$fix(F) = \bigcup_{n \in \boldsymbol{N}} F^n(\bot)$$

Partial order

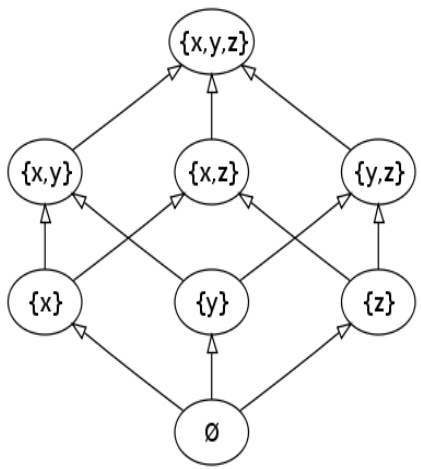
Partial order

Let a set P and a binary relation $\leq \subseteq P \times P$. Relation $\leq$ is a partial order (and $(P, \leq)$ is called a partially ordered set or poset) if and only if:

- $\leq$ is reflexive: $\forall p \in P, p \leq p$
- $\leq$ is transitive: $\forall p, q, r \in P, p \leq q \& q \leq r \implies p \leq r$
- $\leq$ is anti-symmetric: $\forall p, q \in P, p \leq q \& q \leq p \implies p = q$

Ex.: powerset (=set of all subsets of S) with inclusion order $(\wp(S), \subseteq)$

Classical representation as Hasse diagram:



Upper bounds, Lattice, Completeness

Upper/lower bounds

- $p \in P$ is an upper bound of $X \subseteq P$ if $\forall q \in X, q \leq p$
- p is a (the!) least upper bound (lub, sup, denoted $\bigcup X$) if:
  - p is an upper bound of X
  - for all upper bounds q of X, $p \leq q$
- similarly, lower bound and greatest lower bound (glb, inf, denoted $\bigcap X$)

Lattice

- A lattice is a partial order P admitting a lub and a glb for all $X \subseteq P$ containing two elements (hence for any non empty finite set X)
- A complete partial order (cpo) is a partial order P where all ω-chains $p_0 \leq p_1 \leq \ldots \leq p_n \leq \ldots$ of P admit a lub
- In general, we suppose that a cpo also has a minimal element, denoted $\bot$
- A lattice is a complete lattice if all subsets admit a lub (and hence a glb).

Example: in $(\wp(S), \subseteq)$, all X admit a lub (the classical set union) and a glb (the classical set intersection). It is a complete lattice ($\bot = \emptyset$, $\top = S$).

Solution of semantic equations: example

We want the least fixpoint of the system $X = F(X)$ given by

$$F\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} \{-100, \ldots, 50\} \\ x_1 \cup x_4 \\ ]-\infty, 99] \cap x_2 \\ x_3 + 1 \\ [100, +\infty[\cap x_2 \end{pmatrix} = \begin{pmatrix} F_1(x_1, \ldots, x_5) \\ F_2(x_1, \ldots, x_5) \\ \ldots \\ \ldots \\ F_5(x_1, \ldots, x_5) \end{pmatrix}$$

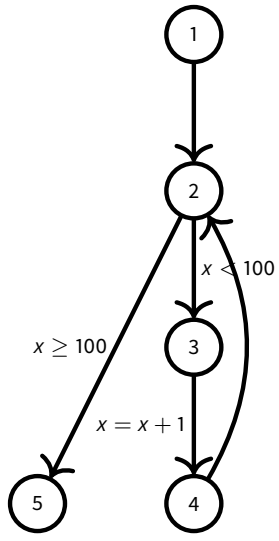
Kleene-like iterations

- classical Kleene iteration $X^0 = \bot, X^1 = F(X^0), \ldots, X^{k+1} = X^k \cup F(X^k)$ is very slow
- we can "locally" iterate on any of the F_is, as long as all F_j are evaluated, potentially, an infinite number of times
- we use here the (classical too) iteration: start from $X^0 = \bot$ and iterate

$$\begin{aligned} x_1^{k+1} &= F_1(x_1^k, \ldots, x_4^k, x_5^k) \\ x_2^{k+1} &= F_2(x_1^{k+1}, x_2^k, \ldots, x_5^k) \\ &\ldots \\ x_5^{k+1} &= F_5(x_1^{k+1}, \ldots, x_4^{k+1}, x_5^k) \end{aligned}$$

Solution of fixpoint computations

```
int x=[-100,50]; //[1]
while /*[2]*/(x<100)
{          // [3]
  x=x+1;   // [4]
}          // [5]
```

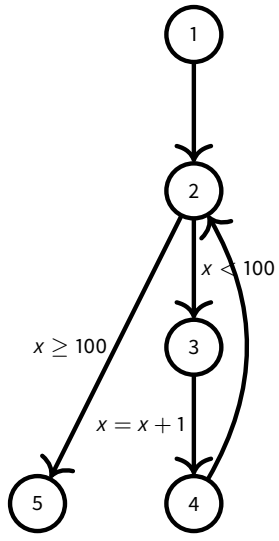


$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_1^0 &= \bot \\
x_2^0 &= \bot \\
x_3^0 &= \bot \\
x_4^0 &= \bot \\
x_5^0 &= \bot
\end{aligned}
$$

Solution of fixpoint computations

```
int x=[-100,50];//[1]
while /*[2]*/(x<100)
{           // [3]
  x=x+1;    // [4]
}           // [5]
```

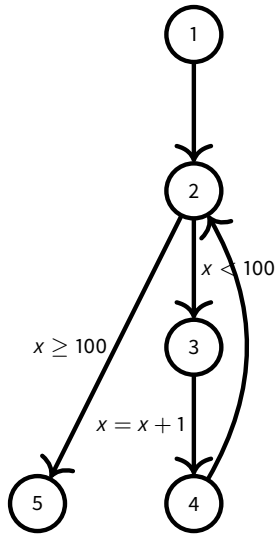


$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_1^0 &= \bot & x_1^1 &= \{-100, \ldots, 50\} \\
x_2^0 &= \bot & x_2^1 &= \{-100, \ldots, 50\} \\
x_3^0 &= \bot & x_3^1 &=]-\infty, 99] \cap \{-100, \ldots, 50\} = \{-100, \ldots, 50\} \\
x_4^0 &= \bot & x_4^1 &= \{-100, \ldots, 50\} + 1 = \{-99, \ldots, 51\} \\
x_5^0 &= \bot & x_5^1 &= [100, +\infty[\cap \{-100, \ldots, 50\} = \bot
\end{aligned}
$$

Solution of fixpoint computations

```
int x=[-100,50];//[1]
while /*[2]*/(x<100)
{          // [3]
  x=x+1;   // [4]
}          // [5]
```

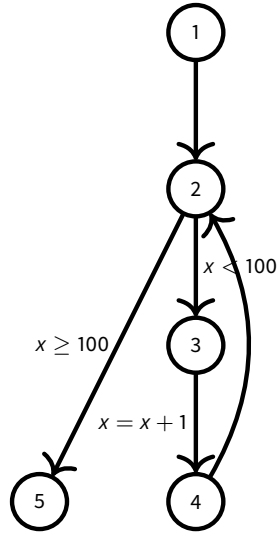


$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} & x_1^2 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} & x_2^2 &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 51\} = \{-100, \ldots, 51\} \\
x_3^1 &= \{-100, \ldots, 50\} & x_3^2 &=]-\infty, 99] \cap \{-100, \ldots, 51\} = \{-100, \ldots, 51\} \\
x_4^1 &= \{-99, \ldots, 51\} & x_4^2 &= \{-100, \ldots, 51\} + 1 = \{-99, \ldots, 52\} \\
x_5^1 &= \bot & x_5^2 &= [100, +\infty[\cap \{-100, \ldots, 51\} = \bot
\end{aligned}
$$

Solution of fixpoint computations

```
int x=[-100,50];//[1]
while /*[2]*/(x<100)
{          // [3]
  x=x+1;   // [4]
}          // [5]
```



$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

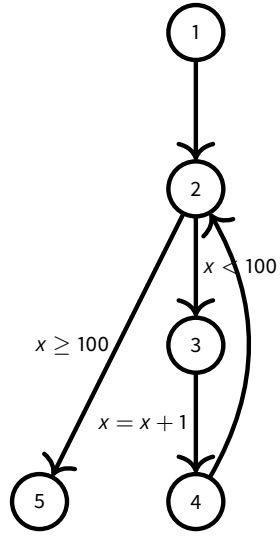
$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} \\
x_3^1 &= \{-100, \ldots, 50\} \\
x_4^1 &= \{-99, \ldots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \ldots, 50\} \\
x_2^2 &= \{-100, \ldots, 51\} \\
x_3^2 &= \{-100, \ldots, 51\} \\
x_4^2 &= \{-99, \ldots, 52\} \\
x_5^2 &= \bot
\end{aligned}
$$

$k < 50$

$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 50 + k\} = \{-100, \ldots, 50 + k\} \\
x_3^{k+1} &=]-\infty, 99] \cap \{-100, \ldots, 50 + k\} = \{-100, \ldots, 50 + k\} \\
x_4^{k+1} &= \{-100, \ldots, 50 + k\} + 1 = \{-99, \ldots, 51 + k\} \\
x_5^{k+1} &= [100, +\infty[\cap \{-100, \ldots, 50 + k\} = \bot
\end{aligned}
$$

Solution of fixpoint computations

```
int x=[-100,50];//[1]
while /*[2]*/(x<100)
{          // [3]
  x=x+1;   // [4]
}          // [5]
```



$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

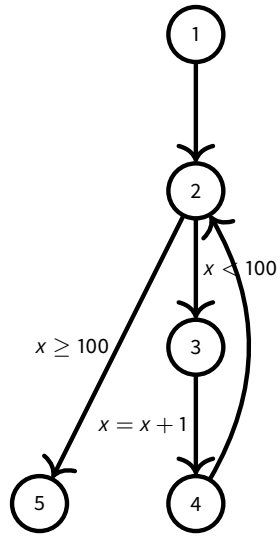
$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} \\
x_3^1 &= \{-100, \ldots, 50\} \\
x_4^1 &= \{-99, \ldots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \ldots, 50\} \\
x_2^2 &= \{-100, \ldots, 51\} \\
x_3^2 &= \{-100, \ldots, 51\} \\
x_4^2 &= \{-99, \ldots, 52\} \\
x_5^2 &= \bot
\end{aligned}
$$

$k < 50$

$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_3^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_4^{k+1} &= \{-99, \ldots, 51 + k\} \\
x_5^{k+1} &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^{51} &= \{-100, \ldots, 50\} \\
x_2^{51} &= \{-100, \ldots, 50\} \cup \{-99, \ldots, 100\} = \{-100, \ldots, 100\} \\
x_3^{51} &=]-\infty, 99] \cap \{-100, \ldots, 100\} = \{-100, \ldots, 99\} \\
x_4^{51} &= \{-100, \ldots, 99\} + 1 = \{-99, \ldots, 100\} \\
x_5^{51} &= [100, +\infty[\cap \{-100, \ldots, 100\} = 100
\end{aligned}
$$

Solution of fixpoint computations

```
int x=[-100,50]; //[1]
while /*[2]*/(x<100)
{          // [3]
  x=x+1;   // [4]
}          // [5]
```



$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= x_1^{k+1} \cup x_4^k \\
x_3^{k+1} &=]-\infty, 99] \cap x_2^{k+1} \\
x_4^{k+1} &= x_3^{k+1} + 1 \\
x_5^{k+1} &= [100, +\infty[\cap x_2^{k+1}
\end{aligned}
$$

$$
\begin{aligned}
x_1^1 &= \{-100, \ldots, 50\} \\
x_2^1 &= \{-100, \ldots, 50\} \\
x_3^1 &= \{-100, \ldots, 50\} \\
x_4^1 &= \{-99, \ldots, 51\} \\
x_5^1 &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^2 &= \{-100, \ldots, 50\} \\
x_2^2 &= \{-100, \ldots, 51\} \\
x_3^2 &= \{-100, \ldots, 51\} \\
x_4^2 &= \{-99, \ldots, 52\} \\
x_5^2 &= \bot
\end{aligned}
$$

$k < 50$

$$
\begin{aligned}
x_1^{k+1} &= \{-100, \ldots, 50\} \\
x_2^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_3^{k+1} &= \{-100, \ldots, 50 + k\} \\
x_4^{k+1} &= \{-99, \ldots, 51 + k\} \\
x_5^{k+1} &= \bot
\end{aligned}
\qquad
\begin{aligned}
x_1^{51} &= \{-100, \ldots, 50\} \\
x_2^{51} &= \{-100, \ldots, 100\} \\
x_3^{51} &= \{-100, \ldots, 99\} \\
x_4^{51} &= \{-99, \ldots, 100\} \\
x_5^{51} &= 100
\end{aligned}
$$

$k \geq 51, X^k = X^{51}$

Partial conclusion

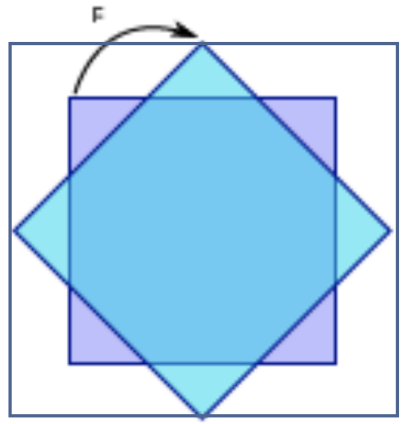
- We were lucky, there is no reason that we can solve these semantic equations in general in finite time
- $\longrightarrow$ Abstract the semantics by elements that are computer representable, with tractable transfer functions
- $\longrightarrow$ Use widening operators that guarantee reaching in finite time a post-fixpoint (such that $F(X) \subseteq X$)

Synthesis of an abstract inductive invariant, informally

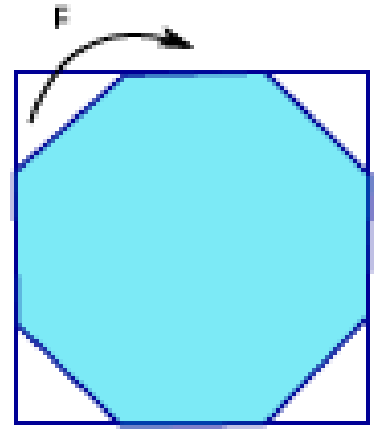
- the program semantics is abstracted by a transition relation or discrete dynamical system $F^\sharp$ that covers all possible behaviors, and can be efficiently computed
- we collect and gather all states that can be reached after zero, one, or any number of iteration steps of $F^\sharp$ = iterate $F^\sharp$ until fixpoint: Kleene iteration
- the least fixpoint when it exists is the strongest inductive invariant (S such that $I \subseteq S$ and $F^\sharp(S) \subseteq S$) of the abstract system
- the abstract fixpoint is a sound abstraction of the concrete fixpoint

```
x := Input[-1,1];
y := Input[-1,1];
while (true) {
  x1 = sqrt(2)/2.*(x+y);
  y1 = sqrt(2)/2.*(x-y);
  x := x1; y:=y1;
}
```

$x, y \in [-2, 2]$ loop invariant ?



no box is an inductive invariant



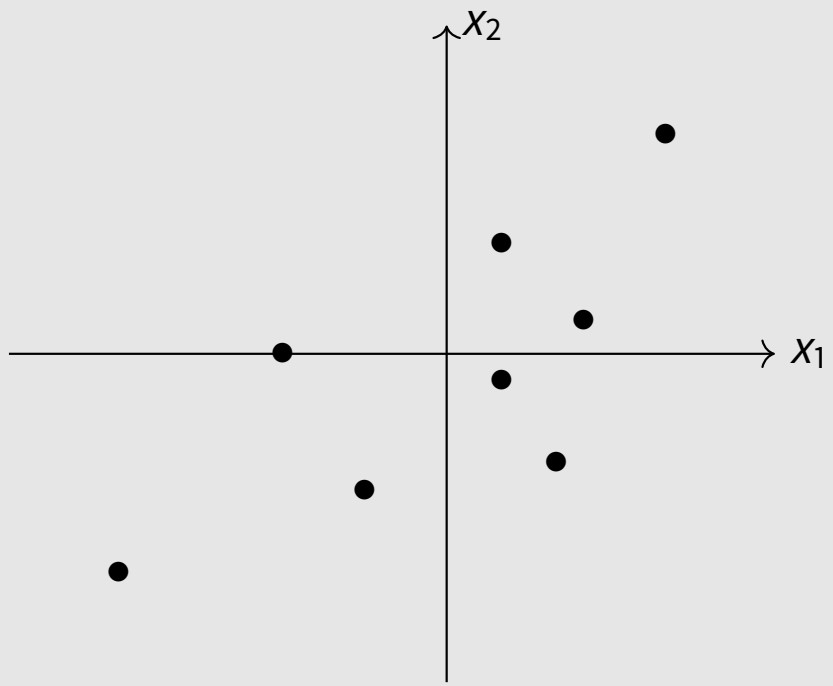
but we can find an inductive octagon

Art of abstract interpretation= define a good general-purpose abstraction!

Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

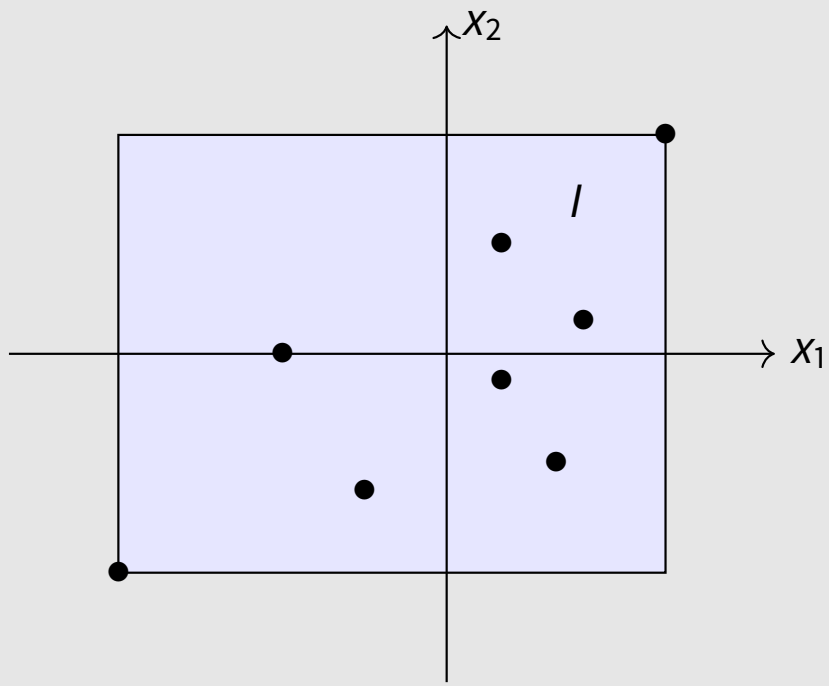
Example (Concrete points,)



Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

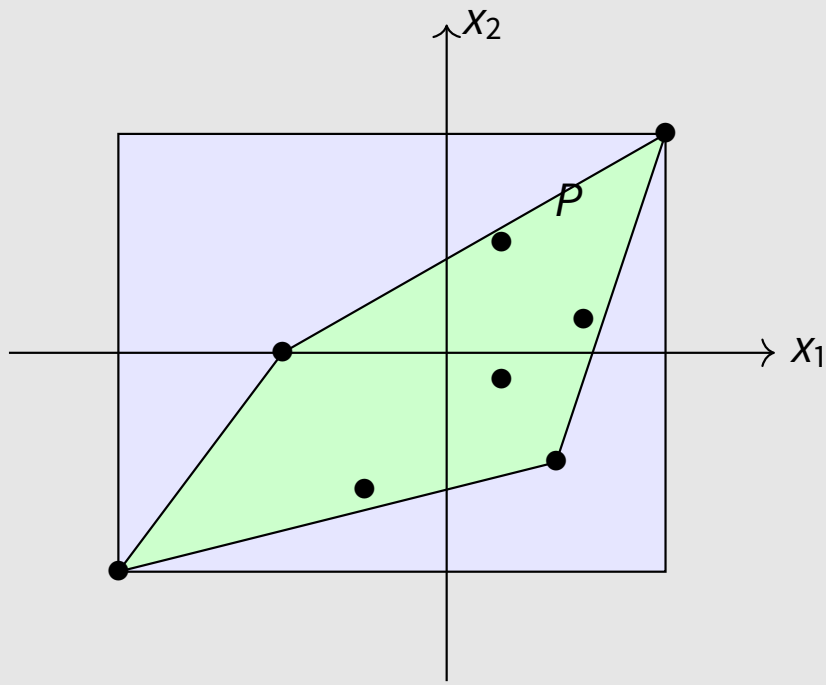
Example (Concrete points, Interval)



Numerical abstract domains

A choice of abstract predicates, that allow an efficient (abstract) representation of sets of (concrete) points, and efficient abstract transfer functions for arithmetic operations.

Example (Concrete points, Interval and Polyhedral abstractions)



Abstract Interpretation (Cousot & Cousot 77)

Abstract interpretation: a theory of semantics approximation

- The simple/elegant framework relies on Galois connections between complete lattices (abstraction by intervals for instance)
- Weaker structures often used in practice

Galois connections

- Let $\mathcal{C}$ be a complete lattice of concrete properties: (e.g. $(\wp(\boldsymbol{Z}), \subseteq)$)
- Let $\mathcal{A}$ be a complete lattice of abstract properties: (e.g. $(S, \sqsubseteq)$)
- $\alpha : \mathcal{C} \to \mathcal{A}$ (abstraction) and $\gamma : \mathcal{A} \to \mathcal{C}$ (concretisation) two monotonic functions (we could forget this hypothesis) such that

$$\alpha(x) \leq_{\mathcal{A}} y \Leftrightarrow x \leq_{\mathcal{C}} \gamma(y)$$

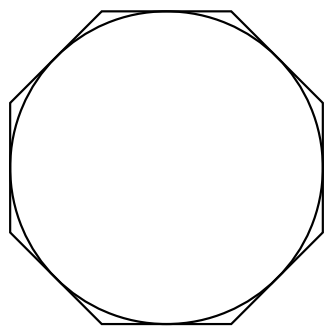
Abstract states over-approximate sets of concrete states: we will reason in the abstract to deduce properties true on the concrete executions

Abstraction

- The (best) abstraction function α gives the most precise abstract value that soundly represents a given concrete property
- The concretization function γ gives the semantics of abstract values, in terms of concrete properties: it maps an abstract value *a* to the largest set of concrete values that satisfy *a*

Not all abstractions are Galois connections

- There can be no α
- (Example: the lattice of polyhedra is not complete: strictly increasing chains with limit not a polyhedron (no best abstraction of a disk)



Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Complete:

$$\bigcup_{i \in I} [a_i, b_i] = [inf_{i \in I}\, a_i, sup_{i \in I}\, b_i]$$

Example: lattice of intervals

- Intervals $[a, b]$ with bounds in $\mathbb{R}$ with $-\infty$ and $+\infty$
- Smallest element $\bot$ identified with all $[a, b]$ with $a > b$
- Greatest element $\top$ identified with $[-\infty, +\infty]$
- Partial order : $[a, b] \subseteq [c, d] \iff a \geq c$ and $b \leq d$

Lattice:

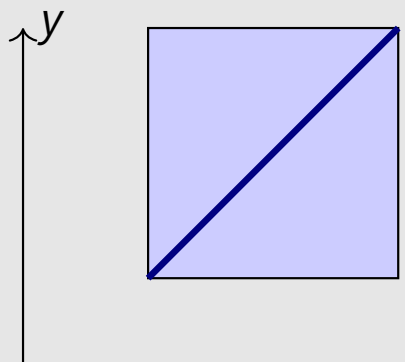
- Sup : $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$
- Inf : $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$

Complete:

$$\bigcup_{i \in I} [a_i, b_i] = [inf_{i \in I}\, a_i, sup_{i \in I}\, b_i]$$

Intervals are a non-relational abstraction: we forget relations between variables

```
int x,y; x=2+rand(0,2);
y=x; x-x = ?
```



Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{rcccc} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & \end{array}$$

Concretisation

$$\begin{array}{rcccc} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a, b] & \to & \end{array}$$

(a and b are potentially infinite)

Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{rcccl} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & [\mathit{inf}\ S, \mathit{sup}\ S] \end{array}$$

(the *inf* and *sup* are taken in $\mathbb{R} \cup \{-\infty, \infty\}$)

Concretisation

$$\begin{array}{rcccl} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a, b] & \to & \end{array}$$

(a and b are potentially infinite)

Galois connection between intervals and sets of reals

Abstraction

$$\begin{array}{rcccl} \alpha & : & \wp(\mathbb{R}) & \to & \mathcal{I} \\ & & S & \to & [\mathit{inf}\ S, \mathit{sup}\ S] \end{array}$$

(the *inf* and *sup* are taken in $\mathbb{R} \cup \{-\infty, \infty\}$)

Concretisation

$$\begin{array}{rcccl} \gamma & : & \mathcal{I} & \to & \wp(\mathbb{R}) \\ & & [a, b] & \to & \{x \in \mathbb{R} \mid a \leq x \leq b\} \end{array}$$

(a and b are potentially infinite)

Abstract transformers

Best abstract transformer of a transfer function F

For all concrete functionals $F : \mathcal{C} \to \mathcal{C}$, we define an abstract functional $F^\sharp : \mathcal{A} \to \mathcal{A}$ by

$$F^\sharp(y) = \alpha \circ F \circ \gamma(y)$$

It is the best possible abstraction of F.

Sound abstract transformer

We can use any over-approximation F' such that $F^\sharp(y) \leq_{\mathcal{A}} F'(y)$:

- must always be conservative (not lose any behavior)
- should provide a good balance between cost and precision
- monotonic to use Kleene iteration for fixpoint computation

Transfer functions for arithmetic expressions

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

Transfer functions for arithmetic expressions

Determining the best interval addition

- Let $+ : \wp(\mathbb{R}) \times \wp(\mathbb{R}) \to \wp(\mathbb{R})$ be the standard, real number addition, lifted onto sets of real numbers
- We want to find its best abstraction on intervals

$$\oplus : \mathcal{I} \to \mathcal{I}$$

We compute:

$$\begin{aligned}
[a,b] \oplus [c,d] &= \alpha\left(\gamma\left([a,b]\right) + \gamma\left([c,d]\right)\right) \\
&= \alpha\left(\{x \mid a \leq x \leq b\} + \{y \mid c \leq y \leq d\}\right) \\
&= \alpha\left(\{x + y \mid a \leq x \leq b,\ c \leq y \leq d\}\right) \\
&= \alpha\left(\{x + y \mid a + c \leq x \leq b + d\}\right) \\
&= [a + c, b + d]
\end{aligned}$$

(+ is extended to infinite numbers)

Transfer functions for conditionals

- $[\![true]\!]\sigma^\sharp = \sigma^\sharp$
- $[\![false]\!]\sigma^\sharp = \bot$
- $[\![x = y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap \sigma^\sharp(y) & \text{if } z = x, y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- $[\![x <= y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap]-\infty, sup\ \sigma^\sharp(y)] & \text{if } z = x \\ \sigma^\sharp(y) \cap [inf\ \sigma^\sharp(x), \infty[& \text{if } z = y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- etc.

Example

What will the interval analysis give in the following program? (detail local invariants)

```
int i = [0,200];
int j = [100,150];
if (i < j)
  x = j - i;
else
  x = i - j;
```

Transfer functions for conditionals

- $[\![true]\!]\sigma^\sharp = \sigma^\sharp$
- $[\![false]\!]\sigma^\sharp = \bot$
- $[\![x = y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap \sigma^\sharp(y) & \text{if } z = x, y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- $[\![x <= y]\!]\sigma^\sharp(z) = \begin{cases} \sigma^\sharp(x) \cap]-\infty, sup\ \sigma^\sharp(y)] & \text{if } z = x \\ \sigma^\sharp(y) \cap [inf\ \sigma^\sharp(x), \infty[& \text{if } z = y \\ \sigma^\sharp(z) & \text{if } z \neq x, y \end{cases}$
- etc.

Example

What will the interval analysis give in the following program? (detail local invariants)

```
int i = [0,200];
int j = [100,150];
if (i < j)
  x = j - i;
else
  x = i - j;
```

- in 1st branch, constraint $i < j$:
  $i = [0, 200] \cap]-\infty, 150[= [0, 149]$,
  $j = [100, 150] \cap]0, \infty[= [100, 150]$, $x = [-49, 150]$
- in 2nd branch, constraint $j \leq i$:
  $j = [100, 150] \cap]-\infty, 200] = [100, 150]$,
  $i = [0, 200] \cap [100, \infty[= [100, 200]$, $x = [-50, 100]$

How do we interpret $x + y == 5$?

(Interval) Constraint Propagation for domain contraction

Constraint Satisfaction Problem (CSP)

A continuous CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ a set of variables
- $\mathcal{D} = \{X_1, \ldots, X_n\}$ a set of domains (intervals)
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ a set of constraints over variables $\mathcal{X}$

Example

$\mathcal{X} = \{x_1, x_2, x_3\}$, given in $\mathcal{D} = [1, 4] \times [1, 4] \times [8, 40]$, with constraint $\mathcal{C} = \{x_3 = x_1 x_2\}$

Domain contraction by forward/backward propagation: remove unfeasible points

For each constraint, isolation of each variable, contraction of its domain, and propagation with the new domain to other variables/constraints

- ideally until a fixed point is reached
- but can stop at any step: sound but possibly imprecise

Computation of a gfp under the initial values of the variables: local decreasing iterations

Interval Constraint Propagation: primitive constraints

Example

$\mathcal{X} = \{x_1, x_2, x_3\}$, given in $\mathcal{D} = [1,4] \times [1,4] \times [8,40]$, with constraint $\mathcal{C} = \{x_3 = x_1 x_2\}$

The constraint $x_3 = x_1 x_2$ can be written in 3 ways $x_3 = x_1 x_2$, $x_1 = x_3 / x_2$ and $x_2 = x_3 / x_1$ that can be used as contractors on each variable:
Iterate (in any order) until fixpoint

- $X_1 := (X_3 / X_2) \cap X_1$
- $X_2 := (X_3 / X_1) \cap X_2$
- $X_3 := X_1 X_2 \cap X_3$

First iteration:

- $X_1 := ([8,40]/[1,4]) \cap [1,4] = [2,4]$
- $X_2 := ([8,40]/[2,4]) \cap [1,4] = [2,4]$
- $X_3 := X_1 X_2 \cap [8,40] = [4,16] \cap [8,40] = [8,16]$

Second iteration:

- $X_1 := ([8,40]/[2,4]) \cap [2,4] = [2,4]$
- $X_2 := ([8,40]/[2,4]) \cap [2,4] = [2,4]$
- $X_3 := X_1 X_2 \cap [8,40] = [8,16]$

Stable, fixpoint reached

Interval constraint propagation for complex constraints

- Complex constraints are rewritten to primitive constraints

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

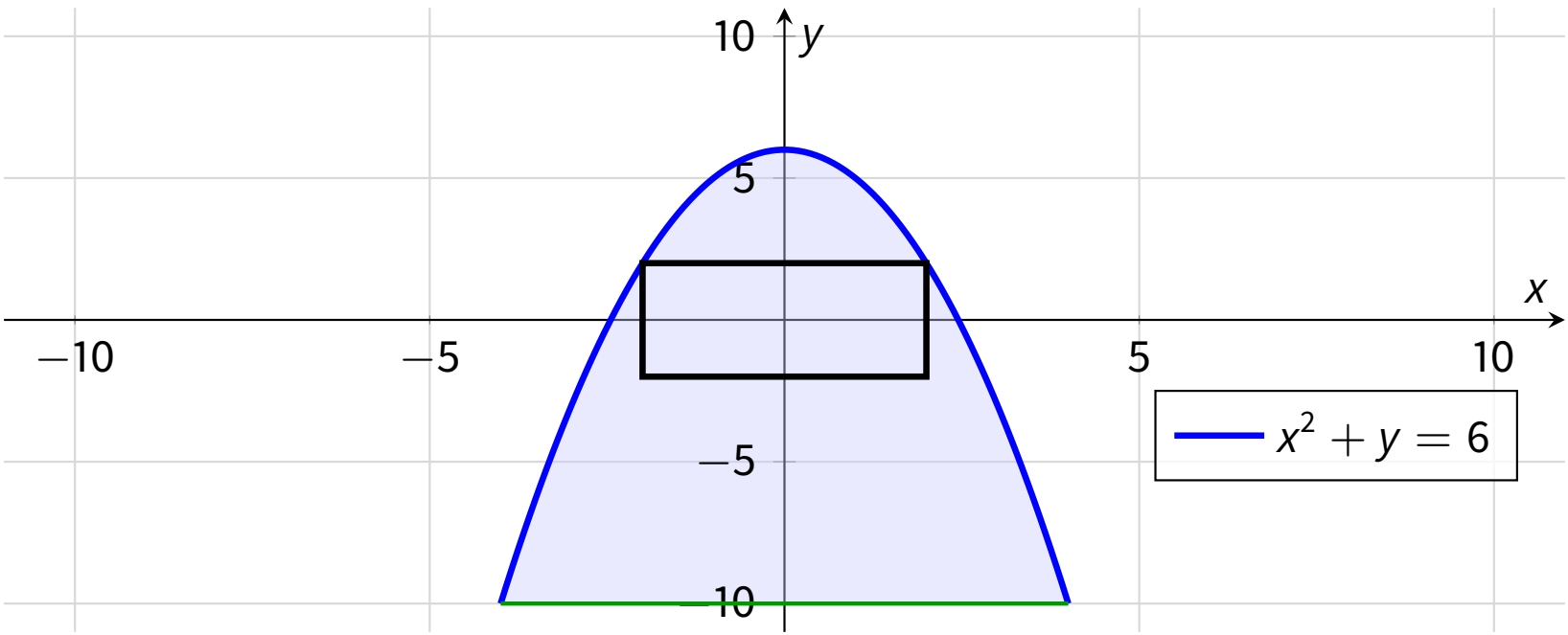
Interval constraint propagation for complex constraints

- Complex constraints are rewritten to primitive constraints

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

- Forward interval propagation yields proof of constraint satisfaction vspace*-0.2cm

$x \in [-2, 2] \wedge y \in [-2, 2] \Rightarrow h_2 = [0, 4] + [-2, 2] \leq 6$: constraint $x^2 + y \leq 6$ is always sati



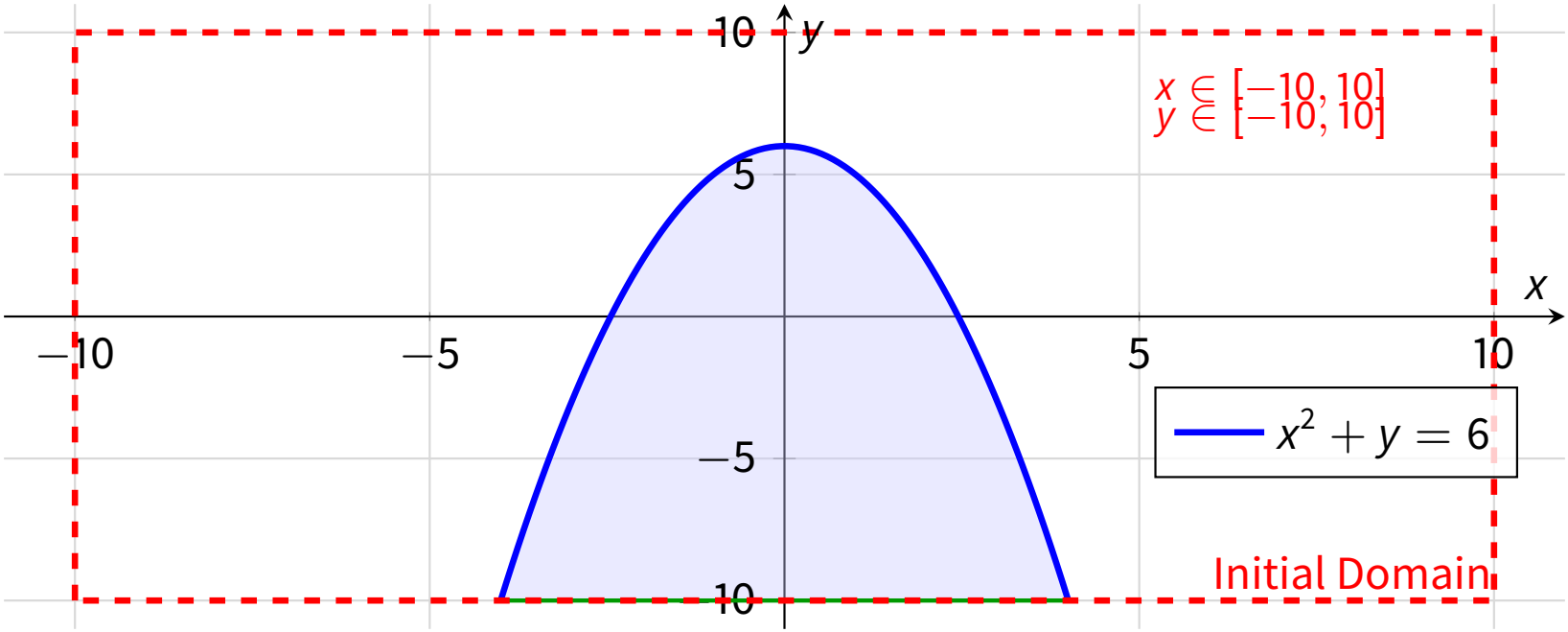
Interval constraint propagation for complex constraints

- Complex constraints are rewritten to primitive constraints

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

- Forward and backward propagation yields contraction of the domain: the domain is contracted given the constraint without removing any feasible solution

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$



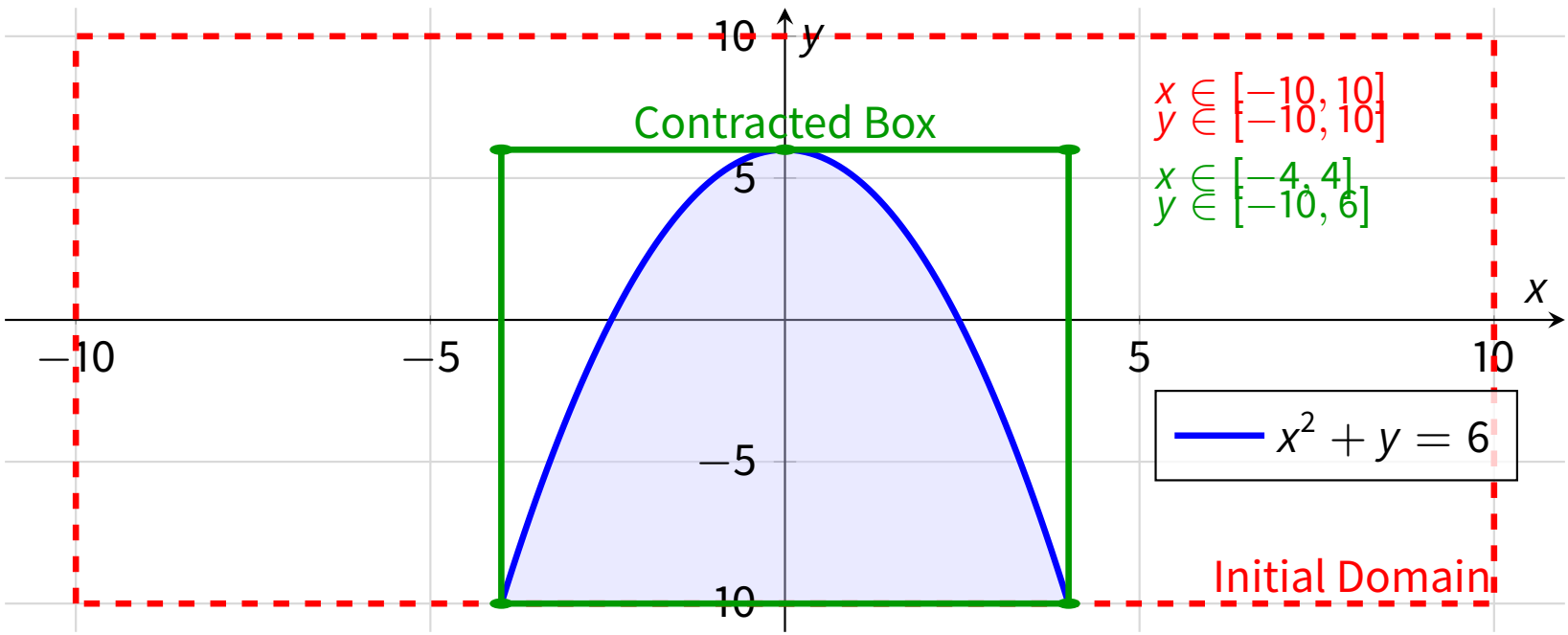
Interval constraint propagation for complex constraints

- Complex constraints are rewritten to primitive constraints

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

- Forward and backward propagation yields contraction of the domain: the domain is contracted given the constraint without removing any feasible solution

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$



Interval constraint propagation for complex constraints

Example

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$

Interval constraint propagation for complex constraints

Example

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$

- Init: $x = [-10, 10], y = [-10, 10]$,
- Forward propagation:
  $h_1 = x^2 = [0, 100], h_2 = h_1 + y = [-10, 110], h_2 := h_2 \cap]-\infty, 6] = [-10, 6]$
- Forward/Backward propagation on $h_2 = h_1 + y$:

  while (not fixpoint)
  $h_1 := h_1 \cap (h_2 - y)$;
  $y := y \cap (h_2 - h_1)$;
  $h_2 := h_2 \cap (h_1 + y)$;

- Conclude by backward propagation on x:

Interval constraint propagation for complex constraints

Example

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$

- Init: $x = [-10, 10], y = [-10, 10]$,
- Forward propagation:
  $h_1 = x^2 = [0, 100], h_2 = h_1 + y = [-10, 110], h_2 := h_2 \cap]-\infty, 6] = [-10, 6]$
- Forward/Backward propagation on $h_2 = h_1 + y$:

  while (not fixpoint)
  $h_1 := h_1 \cap (h_2 - y)$;
  $y := y \cap (h_2 - h_1)$;
  $h_2 := h_2 \cap (h_1 + y)$;

  $$\begin{array}{rcl} h_2 & = & [-10, 110] \cap]-\infty, 6] = [-10, 6] \\ h_1 & = & [0, 100] \cap ([-10, 6] - [-10, 10]) = [0, 100] \cap [-20, 16] = [0, 16] \\ y & = & [-10, 10] \cap ([-10, 6] - [0, 16]) = [-10, 10] \cap [-26, 6] = [-10, 6] \end{array}$$

- Conclude by backward propagation on x:

Interval constraint propagation for complex constraints

Example

$$x^2 + y \leq 6 \rightsquigarrow (h_1 = x^2) \wedge (h_2 = h_1 + y) \wedge (h_2 \leq 6)$$

$$x \in [-10, 10] \wedge y \in [-10, 10] \wedge x^2 + y \leq 6 \Rightarrow x \in [-4, 4] \wedge y \in [-10, 6]$$

- Init: $x = [-10, 10], y = [-10, 10]$,
- Forward propagation:
  $h_1 = x^2 = [0, 100], h_2 = h_1 + y = [-10, 110], h_2 := h_2 \cap]-\infty, 6] = [-10, 6]$
- Forward/Backward propagation on $h_2 = h_1 + y$:

while (not fixpoint)
$h_1 := h_1 \cap (h_2 - y)$;
$y := y \cap (h_2 - h_1)$;
$h_2 := h_2 \cap (h_1 + y)$;

$$\begin{aligned}
h_2 &= [-10, 110] \cap]-\infty, 6] = [-10, 6] \\
h_1 &= [0, 100] \cap ([-10, 6] - [-10, 10]) = [0, 100] \cap [-20, 16] = [0, 16] \\
y &= [-10, 10] \cap ([-10, 6] - [0, 16]) = [-10, 10] \cap [-26, 6] = [-10, 6] \\
\\
h_2 &= [-10, 6] \cap ([0, 16] + [-10, 6]) = [-10, 6] \\
h_1 &= [0, 16] \cap ([-10, 6] - [-10, 6]) = [0, 16] \cap [-16, 16] = [0, 16] \\
y &= [-10, 6]
\end{aligned}$$

- Conclude by backward propagation on x:
  Fixpoint is reached, x such that $x^2 = h_1$ is $x = [-4, 4]$

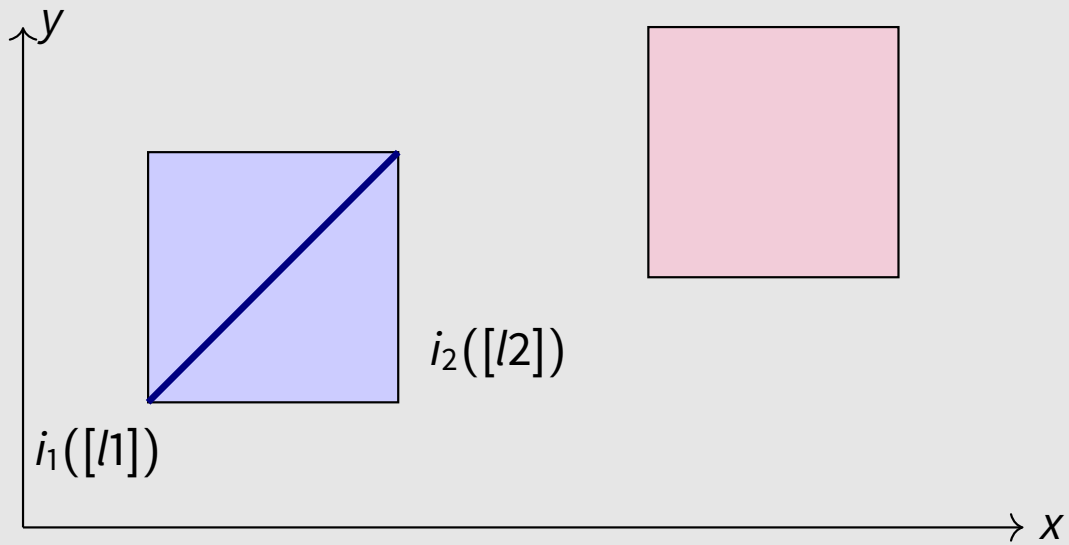
Abstraction and control flow joins

In case of conditional structures

- We must over-approximate the union of abstract values computed in the different branches
- We may lose some precision, but not forget behaviors

Example (Intervals)

```
int x,y;
if (random(0,1)<0.5) {
  x=2+rand(0,2);
  y=x;  [l1]  }
else {   x=6+rand(0,2) ;
  y=3+rand(0,2); [l2]
  }
[l3]
```



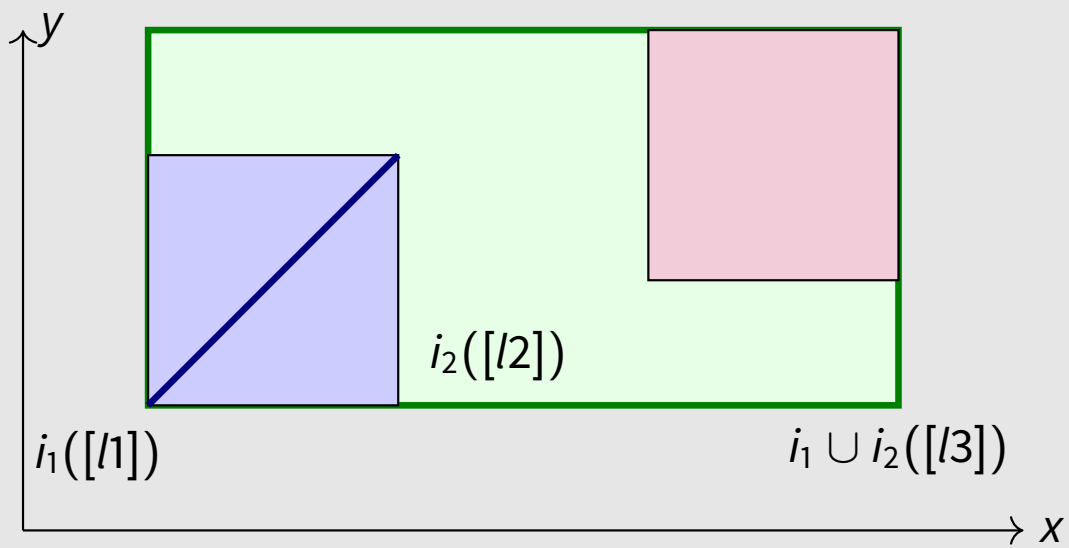
Abstraction and control flow joins

In case of conditional structures

- We must over-approximate the union of abstract values computed in the different branches
- We may lose some precision, but not forget behaviors

Example (Intervals)

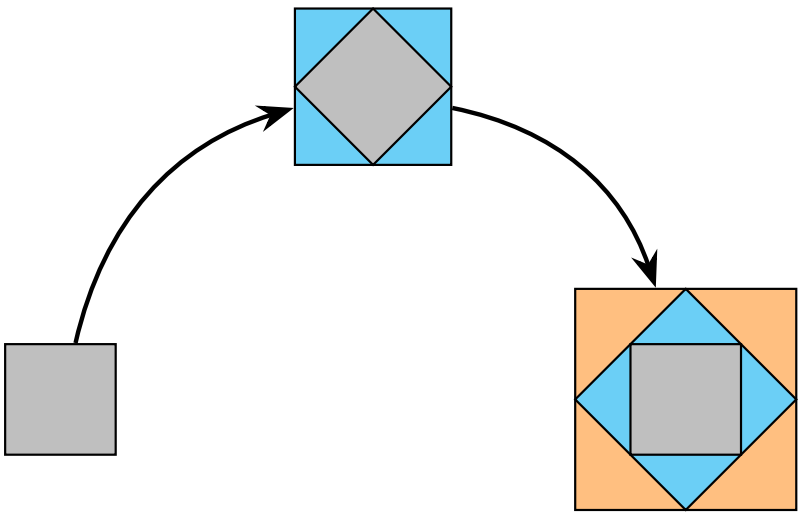
```
int x,y;
if (random(0,1)<0.5) {
  x=2+rand(0,2);
  y=x;  [l1]  }
else {   x=6+rand(0,2) ;
  y=3+rand(0,2); [l2]
  }
[l3]
```



Non relational abstraction: we join abstract values componentwise on x and y

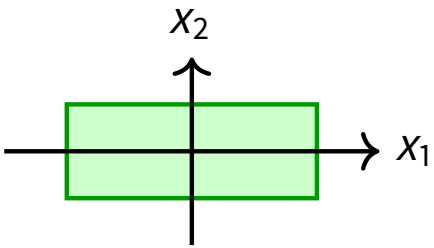
Interval abstraction and wrapping effect

Boxes are non relational: rectangles with sides parallel to the axes cannot capture relations between dimensions

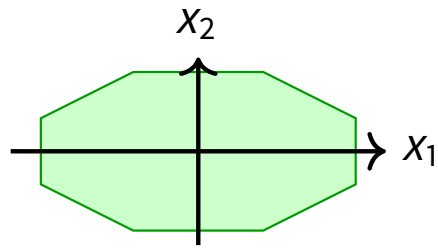


- gray: initial box and its exact image through 2 successive rotations
- cyan: in the center, the best box abstraction of the exact gray image; on the right, its image by the 2nd rotation
- orange: the best box abstraction of the blue image (which is NOT the minimal box enclosing the exact image in gray)

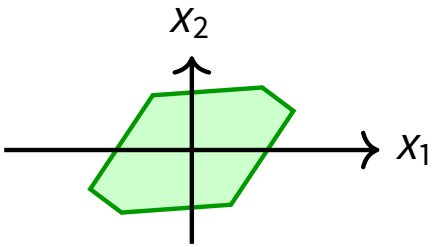
Popular numerical abstract domains



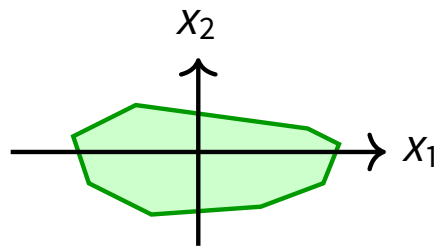
(a) Box: $l_i \leq x_i \leq u_i$



(b) Zone, Octagon: $l_i \leq x_i \leq u_i\,,\ \pm x_i \pm x_j \leq c_i j$



(c) Zonotope: $\hat{x_j} = \alpha_0^j + \sum_{i=1}^{p} g_i^j \epsilon_i$



(d) Polyhedron: $\sum_i a_i x_i \leq c$

- Zones: represent $v_i - v_j \leq c_{ij}$
  [A new numerical abstract domain based on difference-bound matrices. Miné 2001]
- Octagons: zones plus relations $v_i + v_j \leq c_{ij}$ [The octagon abstract domain. A. Miné 2001]
- Polyhedra $\sum_i a_i x_i \leq c$ [Automatic discovery of linear restraints among variables of a program. P. Cousot and N. Halbwachs. POPL 78]

In these 3 cases (see A. Mine's course): explicit relations between variables; often combined with "packings" of variables

Affine Arithmetic (Comba & Stolfi 93)

- A model for self-validated numerical analysis: 1st order guaranteed enclosures to smooth functions
- An improvement of interval arithmetic

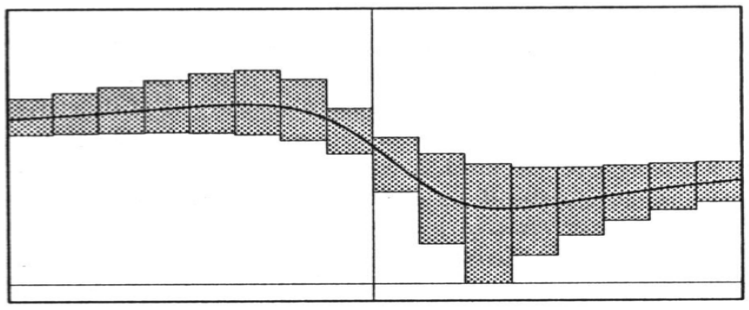


Figure 1: $g(x) = \sqrt{x^2 - x + 1/2}/\sqrt{x^2 + 1/2}$.

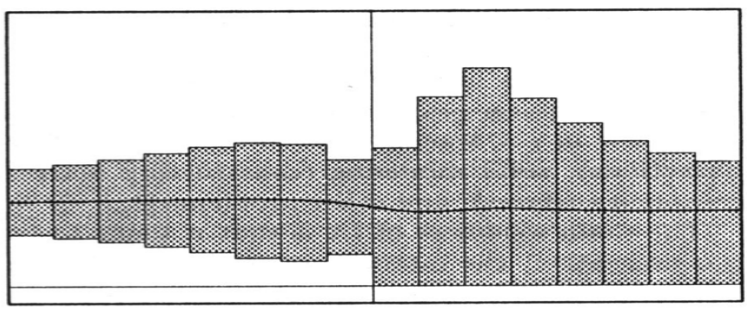


Figure 2: $h(x) = g(g(x))$.

(Standard interval arithmetic.)

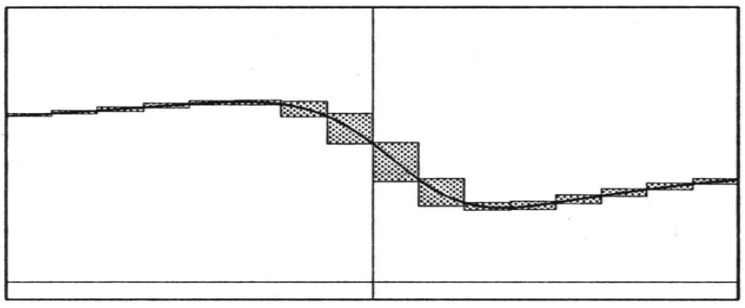


Figure 4: $g(x) = \sqrt{x^2 - x + 1/2}/\sqrt{x^2 + 1/2}$.

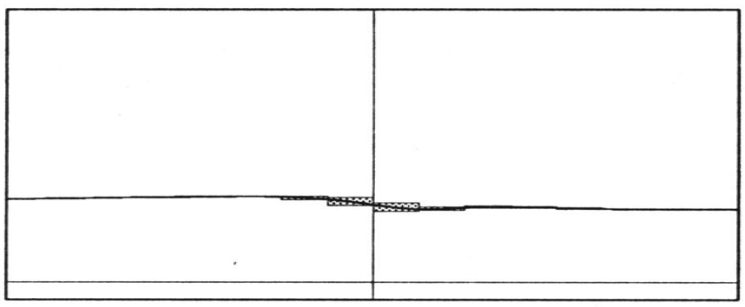


Figure 5: $h(x) = g(g(x))$.

(Affine arithmetic.)

Initially for Computer Graphics: J. L. D. Comba and J. Stolfi (1993), "Affine arithmetic and its applications to computer graphics". Proc. SIBGRAPI'93

Affine forms

Each variable x_k is represented by an Affine Form

$$\hat{x_1} = \alpha_0^1 + \sum_{i=1}^{p} g_i^1 \epsilon_i$$

$$\dots$$

$$\hat{x_n} = \alpha_0^n + \sum_{i=1}^{p} g_i^n \epsilon_i$$

- the ϵ_i are symbolic variables called noise symbols, which range is [-1,1]
- the g_i^k express the magnitude of the corresponding noise in the variable
- sharing ϵ_i between the variables $x_1, \dots, x_n$ expresses implicit dependency

Affine arithmetic (Comba & Stolfi 93)

Affine transformers are exact

Uncertain initial value given in $[a, b]$ for an input variable x introduces a fresh noise symbol ε_i and creates a centered form:

$$\hat{x} = \frac{(a+b)}{2} + \frac{(b-a)}{2}\varepsilon_i$$

Multplication by a constant $\lambda \in \mathbb{R}$ and addition are computed component-wise and exact:

$$\lambda\hat{x} = \lambda x_0 + \lambda x_1\varepsilon_1 + \ldots + \lambda x_p\varepsilon_p$$
$$\hat{x} + \hat{y} = (x_0 + y_0) + (x_1 + y_1)\varepsilon_1 + \ldots + (x_p + y_p)\varepsilon_p$$

Non linear operations : approximate linear form, new noise term bounding the approximation error. For instance multiplication:

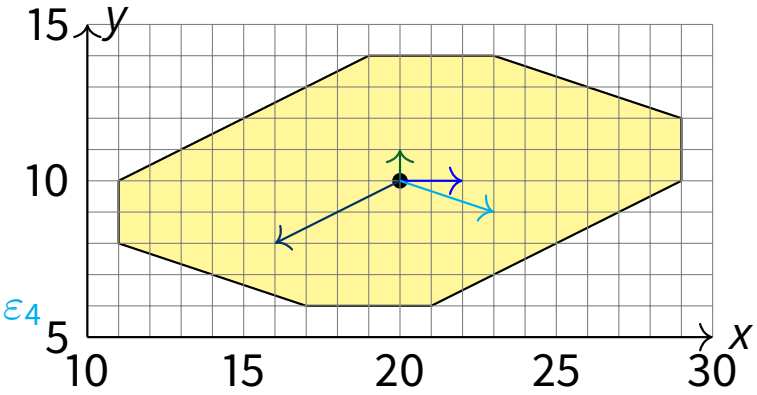
$$\begin{aligned}\hat{x} \times \hat{y} \quad = \quad & (\alpha_0^x\alpha_0^y + \frac{1}{2}\sum_{i=1}^{n}|\alpha_i^x\alpha_i^y|) + \sum_{i=1}^{n}(\alpha_i^x\alpha_0^y + \alpha_i^y\alpha_0^x)\varepsilon_i + \\ & (\frac{1}{2}\sum_{i=1}^{n}|\alpha_i^x\alpha_i^y| + \sum_{1\le i<j\le n}|\alpha_i^x\alpha_j^y + \alpha_j^x\alpha_i^y|)\varepsilon_{n+1}.\end{aligned}$$

Geometrically, a vector of n aff forms with p symb is a zonotope in $\mathbb{R}^n$

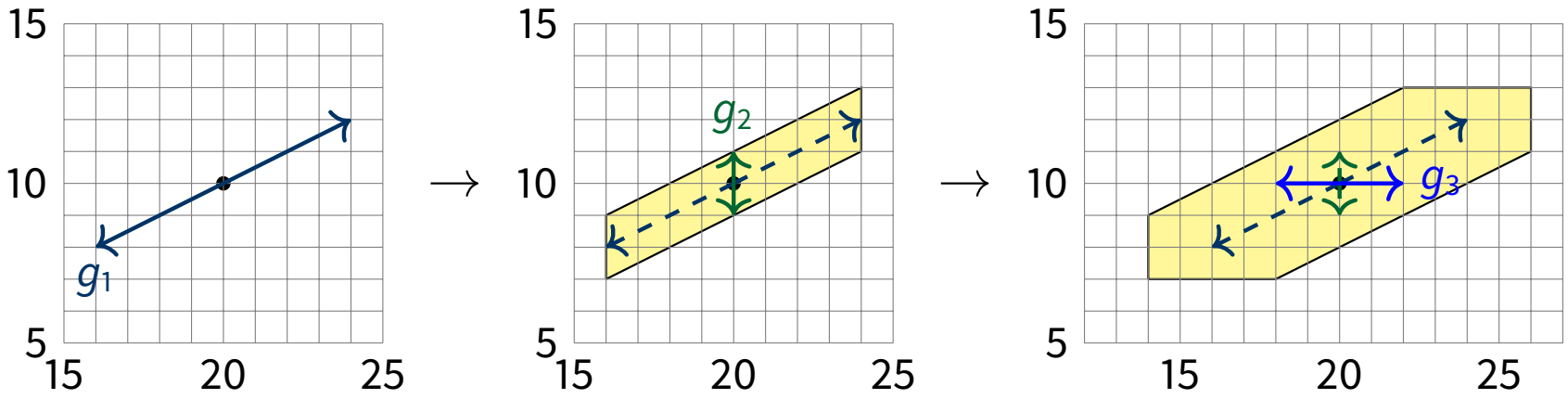
- Zonotope = a Minkowski sum with center c and generators $\{g_1, \ldots, g_p\}$:

$$Z = \{x \in \mathbb{R}^n,\ x = c + \sum_{i=1}^{p} \varepsilon_i g_i,\ -1 \leq \varepsilon_i \leq 1\}$$

$$\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = \begin{pmatrix}20\\ 10\end{pmatrix} + \begin{pmatrix}-4\\ -2\end{pmatrix}\varepsilon_1 + \begin{pmatrix}0\\ 1\end{pmatrix}\varepsilon_2 + \begin{pmatrix}2\\ 0\end{pmatrix}\varepsilon_3 + \begin{pmatrix}3\\ -1\end{pmatrix}\varepsilon_4$$



- Construction in 2D:



Zonotopes

Zonotopes are

- a Minkowski sum of segments

$$Z = \{x \in \mathbb{R}^n,\ x = c + \sum_{i=1}^{p} \varepsilon_i g_i,\ -1 \leq \varepsilon_i \leq 1\}$$

- a center-symmetric polytope which faces are also center-symmetric
- the projection in $\mathbb{R}^n$ of a linear transform of a $p-$dimensional cube

- Zonotopes have been studied in e.g. combinatorial geometry, polyhedral combinatorics,
- Generalization in nD of the 3D zonohedra introduced at the end of the 19th by E. Fedorov, a Russian mathematician and crystallographer.

Example: affine operations

Interpret the following program with interval arithmetic and affine arithmetic:

```
a :=[-1,1] ; b :=[-1,1]
x = 1 + a + 2 * b;
y = 2 - a;
z = x + y - 2 * b;
```

Example: affine operations

Interpret the following program with interval arithmetic and affine arithmetic:

```
a :=[-1,1] ; b :=[-1,1]
x = 1 + a + 2 * b;
y = 2 - a;
z = x + y - 2 * b;
```

- Affine forms for x and y:

$$\hat{x} = 1 + \varepsilon_1 + 2\varepsilon_2 \in [-2, 4]$$

$$\hat{y} = 2 - \varepsilon_1 \in [1, 3],$$

  with $\varepsilon_1, \varepsilon_2 \in [-1, 1]$

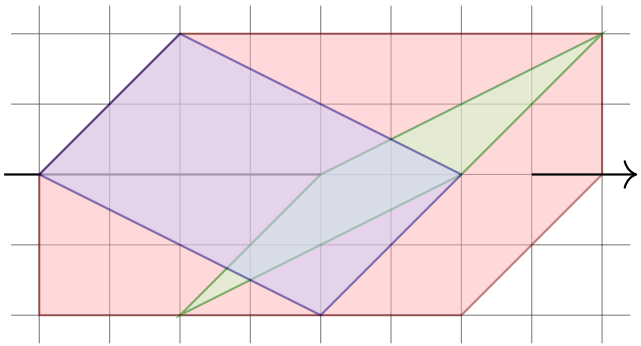
- This representation encodes implicit relations, in particular

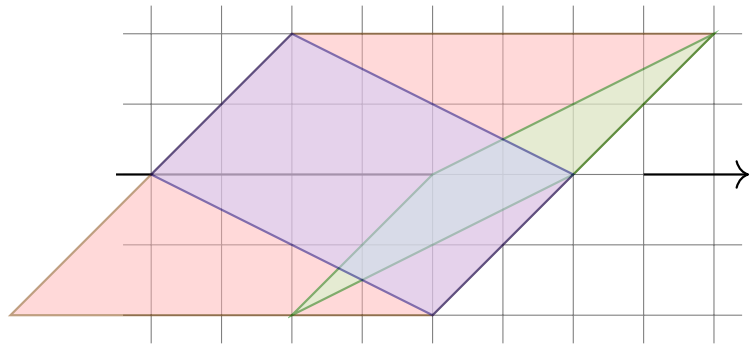
$$\hat{z} = \hat{x} + \hat{y} - 2b = 3$$

- With intervals, $x \in [-2, 4]$, $y \in [-1, 1]$ and $z \in [-3, 9]$

But ... zonotopes with set inclusion ordering do not form a lattice!

No least upper bound of 2 zonotopes: incomparable minimal upper bounds in general





Numerical abstract domain (when not a lattice)

Concretization-based analysis

- Machine-representable abstract values X (affine sets)
- A concretization function γ_f
- A partial order on these abstract values, induced by γ_f: $X \sqsubseteq Y \iff \gamma_f(X) \subseteq \gamma_f(Y)$

Abstract transfer functions

- Arithmetic operations: F is a sound abstraction of f iff

$$\forall x \in \gamma_f(X),\ f(x) \in \gamma_f(F(X))$$

- Set operations: join ($\cup$), meet ($\cap$), widening
  - no least upper bound (possibly several incomparable minimal upper bounds) / greatest lower bound on affine sets
  - Join operator: we want efficient algorithms to compute upper bounds, if possible minimal ones
  - Interpretation of test condition (meet): we need an over-approximation of all lower bounds

and ... hopefully accurate and effective to compute!!!

Affine forms and zonotopes as an abstract domain

Two kinds of noise symbols

- Input noise symbols (ε_i): created by uncertain inputs
- Perturbation noise symbols (η_j): created by uncertainty in analysis

Perturbed affine sets $X = (C^X, P^X)$

$$\begin{pmatrix} \hat{x_1} \\ \hat{x_2} \\ \dots \\ \hat{x_p} \end{pmatrix} = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon_1 \\ \dots \\ \varepsilon_n \end{pmatrix} + {}^tP^X \begin{pmatrix} \eta_1 \\ \eta_2 \\ \dots \\ \eta_m \end{pmatrix}$$

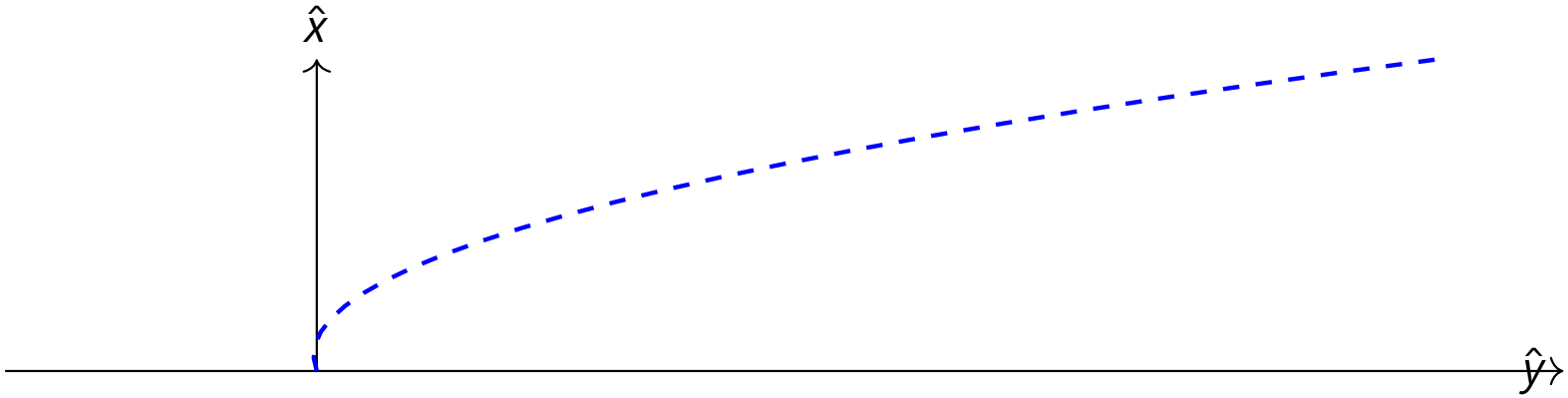
- Central part links the current values of the program variables to the initial values of the input variables (linear functional)
- Perturbation part encodes the uncertainty in the description of values of program variables due to non-linear computations (multiplication, join etc.)

Zonotopes define input-output relations (parameterization by the ε_i)

- Want an order that preserves these input-output relations

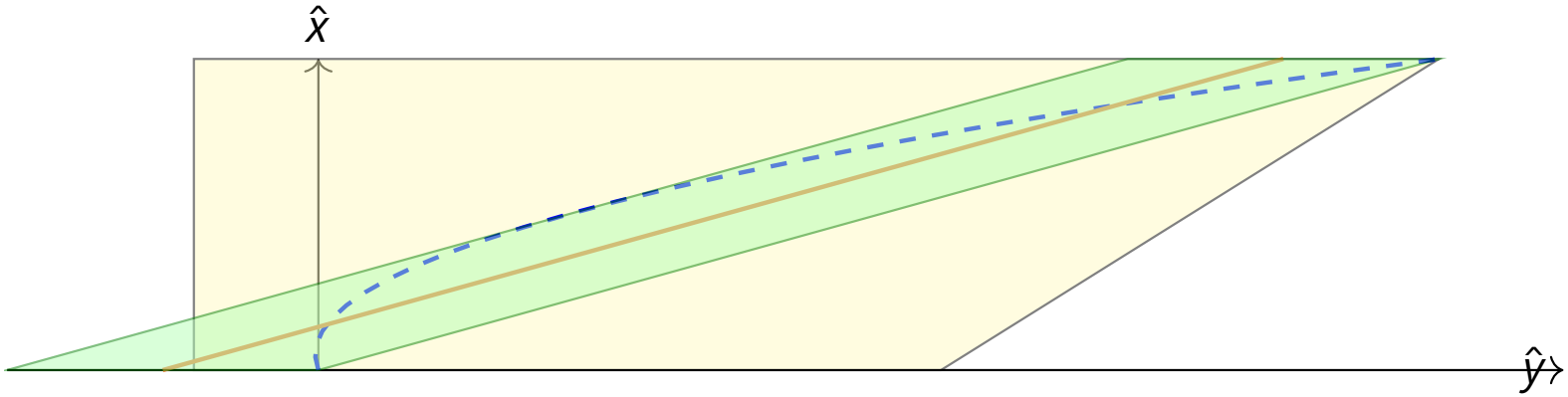
A simple example: functional interpretation

```
real x = [0,10];
real y = x*x - x;
```



A simple example: functional interpretation

```
real x = [0,10];
real y = x*x - x;
```



$$
\begin{aligned}
x &= 5 + 5\varepsilon_1 \\
y &= (5 + 5\varepsilon_1)(5 + 5\varepsilon_1) - 5 - 5\varepsilon_1 \\
&= 20 + 45\varepsilon_1 + 25\varepsilon_1^2 = 20 + 45\varepsilon_1 + 25\,(0.5 + 0.5\eta_1) \\
&= 32.5 + 45\varepsilon_1 + 12.5\eta_1 = -12.5 + 9x + 12.5\eta_1
\end{aligned}
$$

Zonotopes are an efficient abstraction (compared to e.g. the more general polyhedra): geometric shape, but with functional interpretation

- they encode affine correlations between variables in a sparse way.
- they provide first-order enclosures of any smooth function

Functional order relation

Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

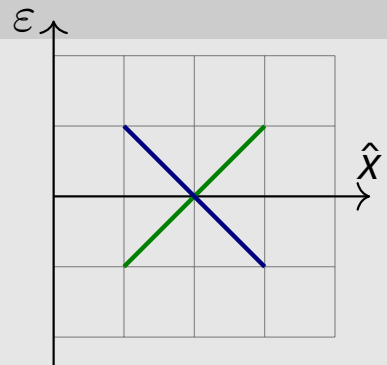
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n\times n} \end{pmatrix} & C^X \\ 0_{m\times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u\in\mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Example

- $x_1 = 2 + \varepsilon_1, x_2 = 2 - \varepsilon_1$
- x_1 and x_2 are incomparable



Functional order relation

Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

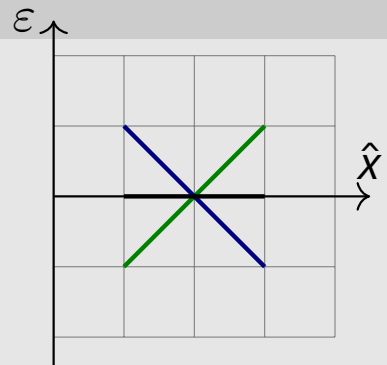
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n\times n} \end{pmatrix} & C^X \\ 0_{m\times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u\in\mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

Example

- $x_1 = 2 + \varepsilon_1, x_2 = 2 - \varepsilon_1$ (geometric concretization [1, 3])
- x_1 and x_2 are incomparable



Functional order relation

Concretization in terms of sets of functions from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$\gamma_f(X) = \left\{ f : \mathbb{R}^n \to \mathbb{R}^p \mid \forall \epsilon \in [-1,1]^n, \exists \eta \in [-1,1]^m, f(\varepsilon) = {}^tC^X \begin{pmatrix} 1 \\ \varepsilon \end{pmatrix} + {}^tP^X \eta \right\}.$$

- Equivalent to the geometric ordering on augmented space $\mathbb{R}^{p+n}$: zonotopes enclosing current values of variables + their initial values ε_i

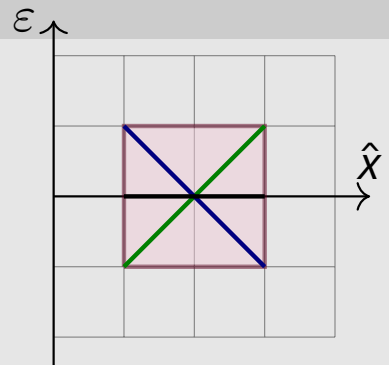
$$\gamma(\tilde{X}) \subseteq \gamma(\tilde{Y}), \text{ where } \tilde{X} = \begin{pmatrix} \begin{pmatrix} 0_n \\ I_{n\times n} \end{pmatrix} & C^X \\ 0_{m\times n} & P^X \end{pmatrix}$$

  - Implies the geometric ordering in $\mathbb{R}^p$
- Computable inclusion test:

$$X \sqsubseteq Y \iff \sup_{u\in\mathbb{R}^p} \left(\|(C^Y - C^X)u\|_1 + \|P^X u\|_1 - \|P^Y u\|_1 \right) \leq 0$$

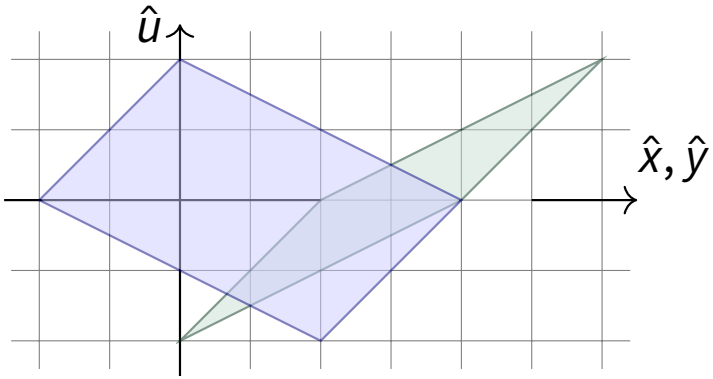
Example

- $x_1 = 2 + \varepsilon_1, x_2 = 2 - \varepsilon_1, x_3 = 2 + \eta_1$ (geometric concretization [1, 3])
- x_1 and x_2 are incomparable , both are included in x_3.



Component-wise upper bound (mub under some conditions)

$$\begin{pmatrix} \hat{x} = 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} \cup \begin{pmatrix} \hat{y} = 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix} = \begin{pmatrix} \hat{x} \cup \hat{y} & = & 2 + \varepsilon_2 + 3\eta_1 \\ \hat{u} & = & 0 + \varepsilon_1 + \varepsilon_2 \end{pmatrix}$$



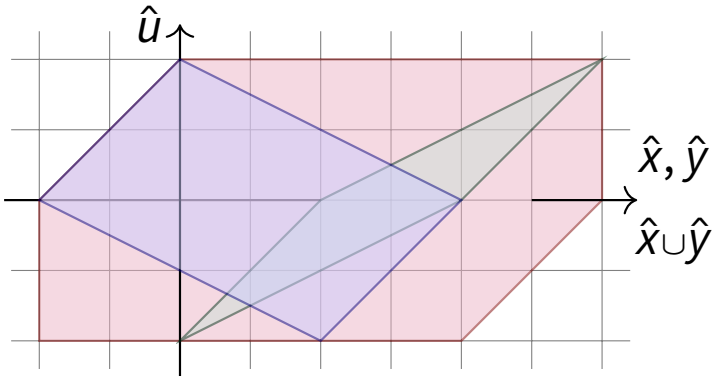
Construction (cost $\mathcal{O}(n \times p)$)

- Keep "minimal common dependencies"

$$z_i = \underset{x_i \wedge y_i \leq r \leq x_i \vee y_i}{\operatorname{argmin}} |r|, \ \forall i \geq 1$$

Component-wise upper bound (mub under some conditions)

$$\left(\begin{array}{l}\hat{x} = 3 + \varepsilon_1 + 2\varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2\end{array}\right) \cup \left(\begin{array}{l}\hat{y} = 1 - 2\varepsilon_1 + \varepsilon_2 \\ \hat{u} = 0 + \varepsilon_1 + \varepsilon_2\end{array}\right) = \left(\begin{array}{lcl}\hat{x} \cup \hat{y} & = & 2 + \varepsilon_2 + 3\eta_1 \\ \hat{u} & = & 0 + \varepsilon_1 + \varepsilon_2\end{array}\right)$$



Construction (cost $\mathcal{O}(n \times p)$)

- Keep "minimal common dependencies"

$$z_i = \underset{x_i \wedge y_i \leq r \leq x_i \vee y_i}{\operatorname{argmin}} |r|, \ \forall i \geq 1$$

- For each dimension, concretization is the interval union of the concretizations: $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$

Fixpoint computation with componentwise join

General result on recursive linear filters, pervasive in embedded programs:

$$x_{k+n+1} = \sum_{i=1}^{n} a_i x_{k+i} + \sum_{j=1}^{n+1} b_j e_{k+j}, \quad e[*] = input(m, M);$$

- Suppose this concrete scheme has bounded outputs (zeros of $x^n - \sum_{i=0}^{n-1} a_{i+1} x^i$ have modules stricty lower than 1).
- Then there exists q such that the Kleene abstract scheme "unfolded modulo q" converges towards a finite over-approximation of the outputs

$$\hat{X}_i = \hat{X}_{i-1} \cup f^q(E_i, \ldots, E_{i-k}, \hat{X}_{i-1}, \ldots, \hat{X}_{i-k})$$

in finite time, potentially with a widening partly losing dependency information

- The abstract scheme is a perturbation (by the join operation) of the concrete scheme
- Uses the stability property of our join operator: for each dimension $\gamma(\hat{x} \cup \hat{y}) = \gamma(\hat{x}) \cup \gamma(\hat{y})$

Transfer functions for conditionals: constrained zonotopes

- Affine forms unchanged (keep correlations), transfer constraint on noise symbols
- Equality constraints: substitution of one noise symbol using the equality
- Inequality constraints: use a contractor to contract the domain of the noise symbols ε_i (or keep the constraints explicitely and use LP)

Example

```
real x = [0,10];
real y = 2*x;
if (y >= 10)
  y = x;
```

- Affine forms before tests: $x = 5 + 5\varepsilon_1, y = 10 + 10\varepsilon_1$
- In the `if` branch $\varepsilon_1 \geq 0$: condition acts on both x and y

Functional interpretation

The conditional leads to a constraint on the noise symbols ε_i restricting to the set of inputs that can lead to an execution satisfying the constraint.

Application: examples of analyzes

Some classical abstractions/analyses

- Run-Time Errors (RTE: division by zero, nil dereference, out of bounds array dereference, overflows etc.)
- Worst-Case Execution Time (WCET)

Next slides: focus on the analysis of numerical properties

- Accuracy of finite precision implementations
- Behaviour of the program (control flow, number of iterations, method error)

Analysis of floating-point programs: a simple example

```c
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.0e11 - 1;
    y = 1.0e11 + 1;
    z = x - y;
    r = 1/z;
    printf("%f %f\n", z, r);
}
```

Analysis of floating-point programs: a simple example

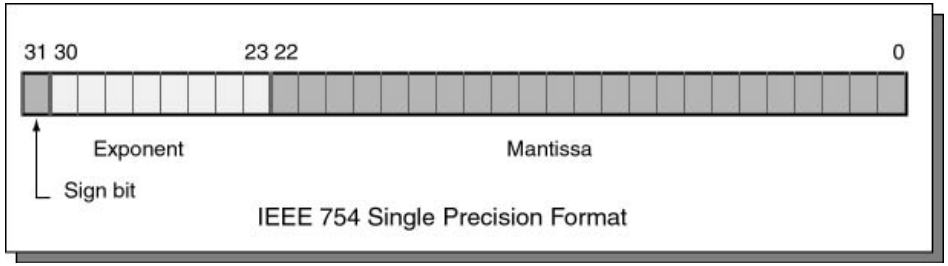
```c
/* float-error.c */
int main () {
    float x, y, z, r;
    x = 1.0e11 - 1;
    y = 1.0e11 + 1;
    z = x - y;
    r = 1/z;
    printf("%f %f\n", z, r);
}
```

> gcc float-error.c

> ./a.out

> 0.000000 inf

Floating-point numbers (IEEE 754 norm)



$f = (-1)^s \, 1.m \, 2^{e-bias}$, with s the sign, m is the mantissa, e the exponent.

- Limited range and precision : potentially inaccurate results or run-time errors
- absorption/cancellation : $1 + 10^{-8} = 1$ in simple precision float
- loss of associativity : $(-1 + 1) + 10^{-8} \neq -1 + (1 + 10^{-8})$
- Elementary rounding error when rounding r^x to $fl(r^x)$:

$$\exists(\delta_r > 0, \delta_a > 0), \ |r^x - fl(r^x)| \leq max(\delta_r |fl(r^x)|, \delta_a)$$

- The f.p. result of arithmetic elementary operations $+, -, \times, /, \sqrt{}$ is the rounded value of the real result
  - allows to systematically bound the error committed at each arithmetic operation
  - basis for an error analysis that bounds and propagates these elementary errors

Concrete semantics underlying the FLUCTUAT analyzer from CEA

- Bound and propagate rounding errors and track their origin:
  - bound floating-point value, but relational analysis only on real values
  - for each variable x, we compute f^x from (r^x, e^x)
  - abstract real value and errors by zonotopes: noise symbols also encode origin

```
float x,y,z;
x = 0.1; // [1]
y = 0.5; // [2]
z = x+y; // [3]
t = x*z; // [4]
```

$$
\begin{aligned}
f^x &= 0.1 + 1.49e^{-9}\,[1] \\
f^y &= 0.5 \\
f^z &= 0.6 + 1.49e^{-9}\,[1] + 2.23e^{-8}\,[3] \\
f^t &= 0.06 + 1.04e^{-9}\,[1] + 2.23e^{-9}\,[3] - 8.94e^{-10}\,[4] - 3.55e^{-17}\,[ho]
\end{aligned}
$$

- Implementation:
  - Accurately bound floating-point errors: convenient to use multiple-precision floating-point numbers: the GNU MPFR library (`https://www.mpfr.org`)
  - Sound finite-precision implementation of affine forms: either use affine forms with (tight) interval coefficients or bound error at each operation

Abstraction in Fluctuat

- Can we prove:
  - that the algorithm computes something close to what is expected ? (method error)
  - that it does so also in finite precision? (finite precision error)
  - that the finite precision behaviour (control flow) of the scheme is satisfying? (problem of unstable tests + convergence for iterative schemes)

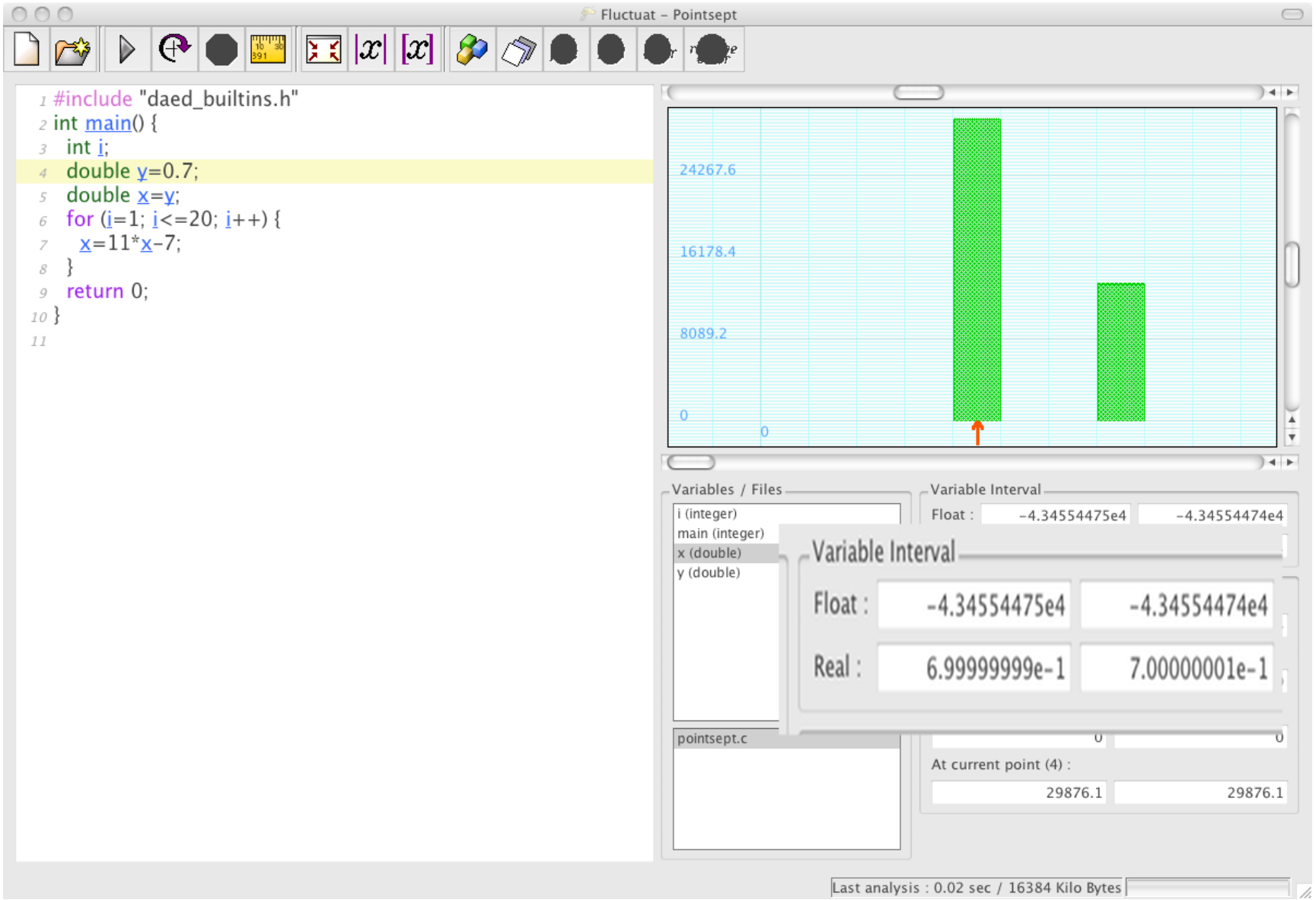
Abstract state:

- For each variable:
  - Interval $\boldsymbol{f}^x = [\underline{f^x}, \overline{f^x}]$ bounds the finite prec value, $(\underline{f^x}, \overline{f^x}) \in \mathbb{F} \times \mathbb{F}$, intersected with

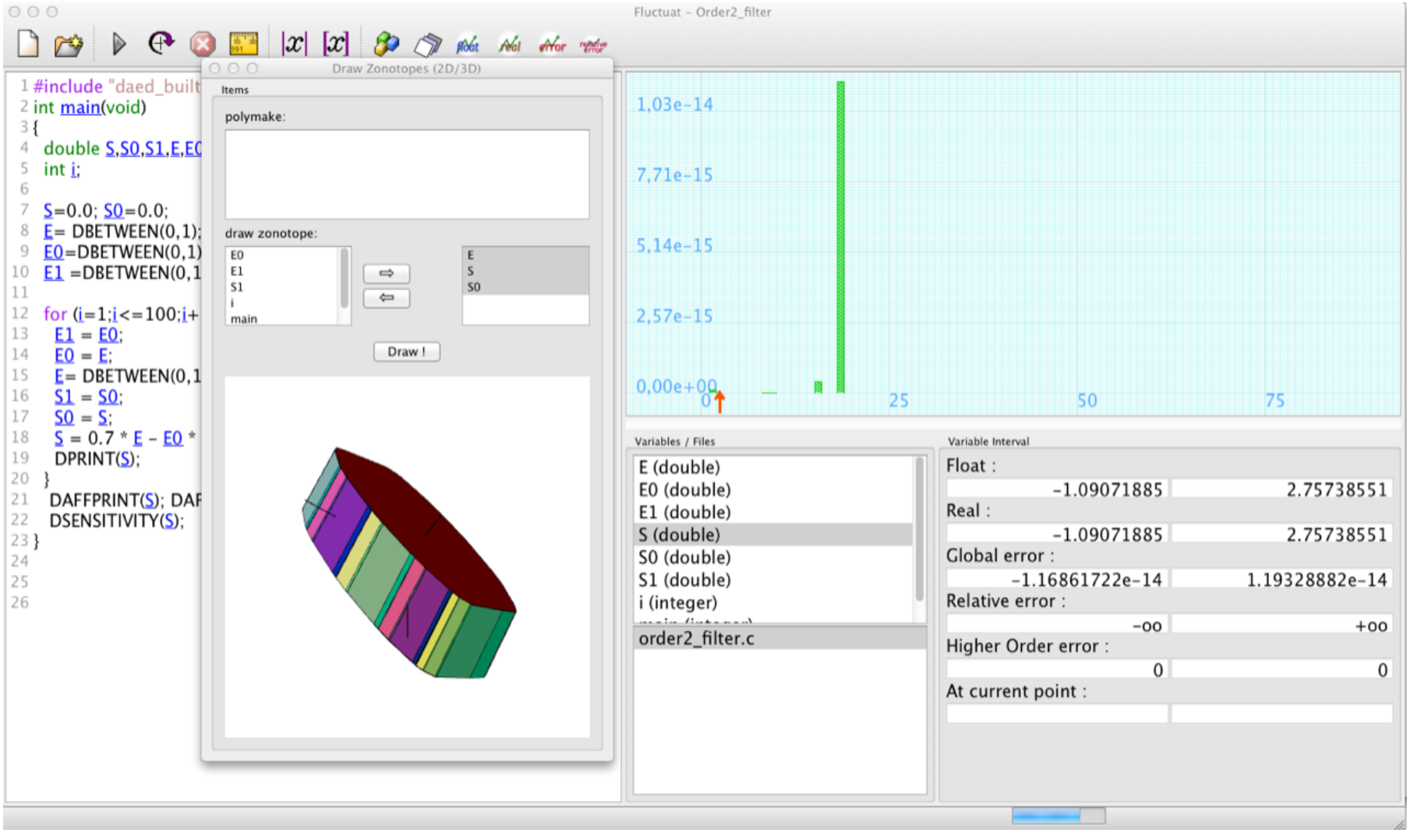
$$f^x = \underbrace{(\alpha_0^x + \bigoplus_i \alpha_i^x \varepsilon_i^r)}_{\text{real value}} + (\underbrace{e_0^x}_{\text{center of the error}} + \underbrace{\bigoplus_l e_l^x \varepsilon_l^e}_{\text{uncertainty on error due to point } l} + \underbrace{\bigoplus_i m_i^x \varepsilon_i^r}_{\text{propag of uncertainty on value at pt } i})$$

- Globally for the state, constraints on noise symbols: interpretation of conditional
  - for finite precision control flow
  - for real control flow
- In case of unstable tests (when finite precision and real control flow may differ), the difference between the abstract values in both branches added as an error term

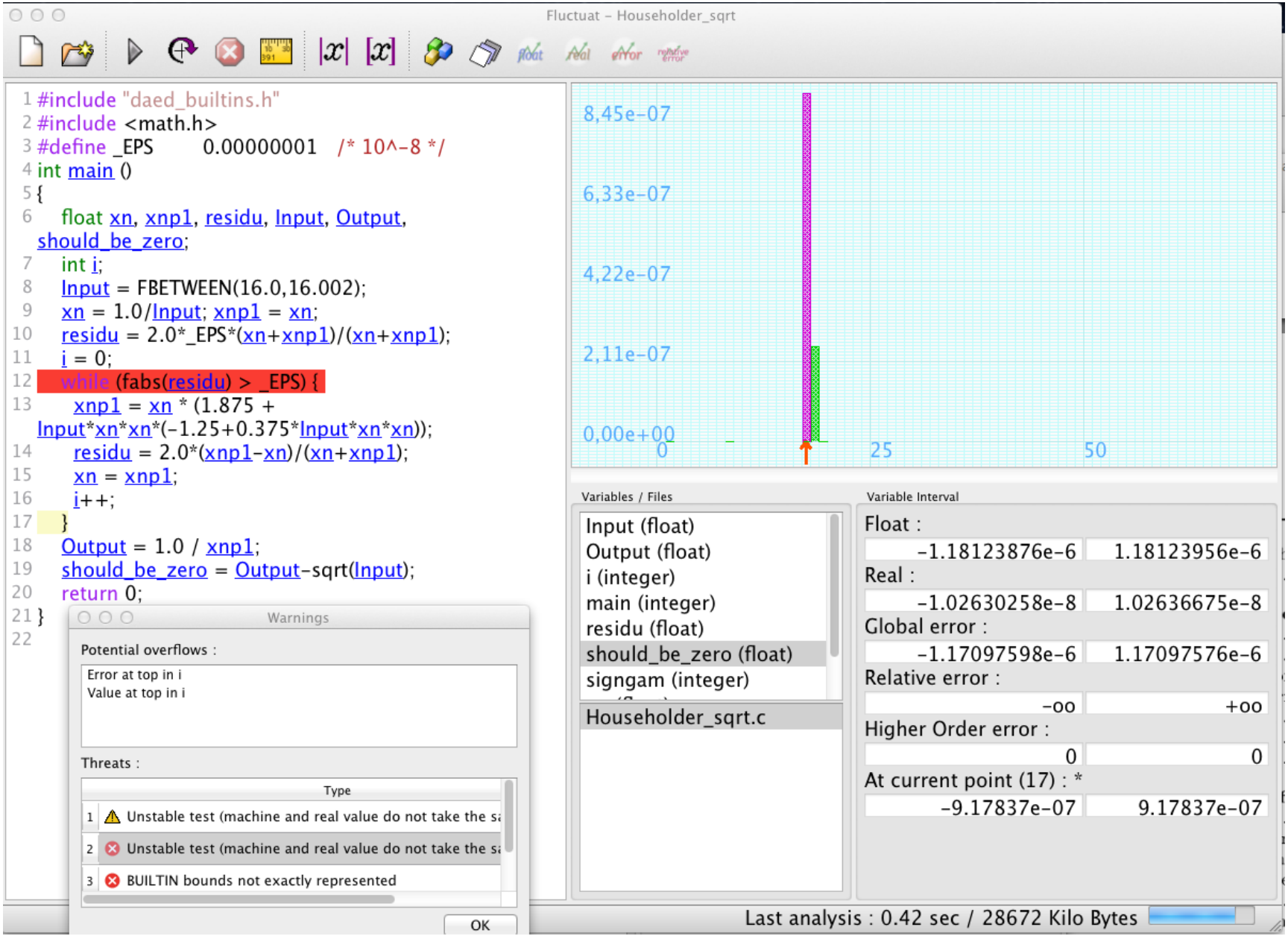
Example (Fluctuat)



Fixpoint for an order 2 recursive filter



Another example (Fluctuat): Householder scheme for square root



Bibliography

Interval analysis:

- R. Moore, Interval Analysis, Prentice Hall, Englewood Cliffs, 1966
- L. Jaulin, M. Kieffer , O. Didrit , É. Walter, Applied Interval Analysis, Springer, 2001

Abstract interpretation:

- A. Mine's MPRI course AISAV: Abstract interpretation: application to verification and static analysis and his lecture notes and references.

The zonotopic abstract domain and application to the analysis of numerical programs:

- E. Goubault and S. Putot, A zonotopic framework for functional abstractions, Formal Methods in System Design, 47(3), pp 302-360, 2016.
- S. Putot, Static analysis of numerical programs and systems, Hab. Thesis, 2012

Next

Assignment for next week: paper reading and presentation

E. Goubault and S. Putot, A zonotopic framework for functional abstractions

- Beware, there is a shorter version on Arxiv, the above link (also on the course's webpage) is the version to consider
- Everybody reads the article, one of you presents the main ideas with slides in approx. 15 minutes
- Follow-up discussion: prepare questions, comments (critical view expected!)
- Each of you will present at least one article during the course
- Who is presenting next week ?

Next week: neural network verification: zonotopes again!