Generic and specific abstract domains for static analysis by abstract interpretation

Antoine Miné

CNRS & École normale supérieure
Paris, France

6ème Rencontres Arithmétiques de l'Informatique Mathématique

19 November 2013

Motivation: a classic example

Maiden flight of the Ariane 5 Launcher, 4 June 1996.

Motivation: a classic example

40s after launch. . .

(cause: overflow during an arithmetic conversion)

Lessons

- software errors can be costly even simple ones
  (Ariane 5 failure estimated at more than $370,000,000$ US$)
- hardware redundancy does not help
  (redundant computers run the same software, the same error)
- testing is not sufficient
  (hardly exhaustive)
- programming in high-level "safe" languages is not sufficient
  (Ariane 5 coded in Ada, with arithmetic exceptions enabled)

Lessons

- software errors can be costly even simple ones
  (Ariane 5 failure estimated at more than $370,000,000$ US$)
- hardware redundancy does not help
  (redundant computers run the same software, the same error)
- testing is not sufficient
  (hardly exhaustive)
- programming in high-level “safe” languages is not sufficient
  (Ariane 5 coded in Ada, with arithmetic exceptions enabled)

$\Longrightarrow$ use **formal methods**
(provide rigorous, mathematical insurance about program behaviors)

Static analysis

Semantic-Based Static Analysis

Infers properties of the dynamic behavior of programs.

- analyzes the source code (not a model)
- soundness: no behavior is missed (full control and data coverage)
- automatic, always terminates
- incomplete due to over-approximations (false alarms)

Applications:

- check simple properties, with low precision requirements
  (optimization in compilers)
- can be used to uncover bugs
  (Ariane 5 bug detected by Polyspace Analyzer, late 1990s)
- can it be used for validation
  (0 false alarm goal; e.g, Astrée specialized analyzer, early 2000s)

Example analysis: inferring numeric invariants

Insertion Sort

```
for i=1 to 99 do
  p := T[i]; j := i+1;
  while j <= 100 and T[j] < p do
    T[j-1] := T[j]; j := j+1;
  end;
  T[j-1] := p;
end;
```

Example analysis: inferring numeric invariants

Interval analysis:

Insertion Sort

```
for i=1 to 99 do
   i ∈ [1,99]
  p := T[i]; j := i+1;
   i ∈ [1,99], j ∈ [2,100]
  while j <= 100 and T[j] < p do
     i ∈ [1,99], j ∈ [2,100]
    T[j-1] := T[j]; j := j+1;
     i ∈ [1,99], j ∈ [3,101]
  end;
   i ∈ [1,99], j ∈ [2,101]
  T[j-1] := p;
end;
```

$\Longrightarrow$ there is no out of bound array access

Example analysis: inferring numeric invariants

Linear inequality analysis:

Insertion Sort

```
for i=1 to 99 do
    i ∈ [1,99]
  p := T[i]; j := i+1;
    i ∈ [1,99], j = i+1
  while j <= 100 and T[j] < p do
      i ∈ [1,99], i+1 ≤ j ≤ 100
    T[j-1] := T[j]; j := j+1;
      i ∈ [1,99], i+2 ≤ j ≤ 101
  end;
    i ∈ [1,99], i+1 ≤ j ≤ 101
  T[j-1] := p;
end;
```

Abstract interpretation

Abstract interpretation: unifying theory of program semantics

[Cousot Cousot 76]

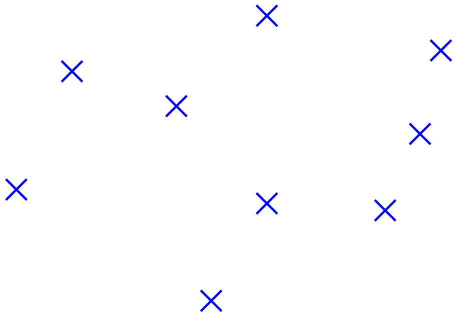
Core principles:

- semantics are linked through abstractions (α, γ)
- abstractions can be composed and reused (abstract domain)
- semantics are expressed as fixpoints (lfp F)
- fixpoints can be approximated by iteration with acceleration (widening $\triangledown$)

Applications:

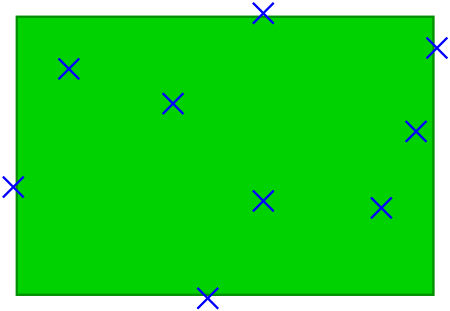
- compare existing semantics and analyses (unifying power)
- derive new semantics by abstraction
  derive computable semantics $\Longrightarrow$ **sound** static analysis

Abstract domain examples



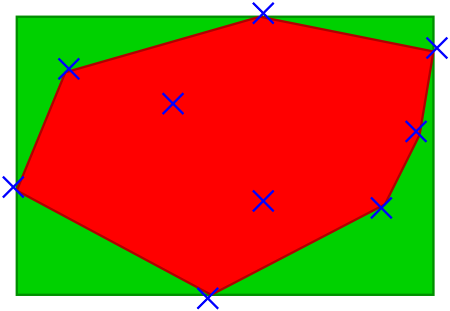
concrete $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$ (not computable)

Abstract domain examples



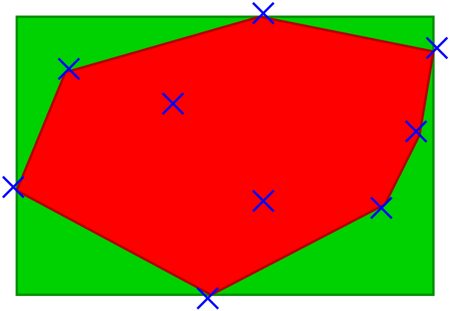
concrete $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$	(not computable)
boxes $\mathcal{D}^\sharp_b$:	$X \in [0;12] \wedge Y \in [0;8]$	(linear cost)

Abstract domain examples



concrete $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$	(not computable)
boxes $\mathcal{D}^\sharp_b$:	$X \in [0;12] \wedge Y \in [0;8]$	(linear cost)
polyhedra $\mathcal{D}^\sharp_p$:	$6X + 11Y \geq 33 \wedge \cdots$	(exponential cost)

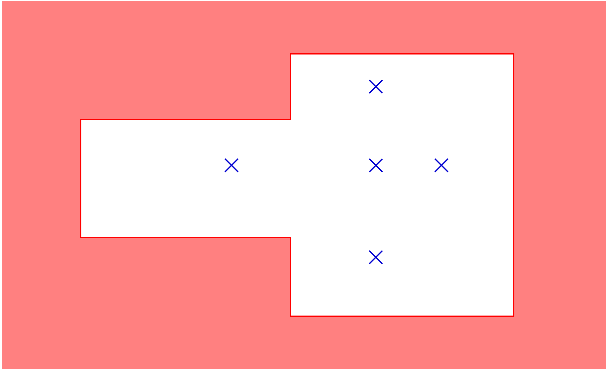
Abstract domain examples



concrete $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$	(not computable)
boxes $\mathcal{D}_b^\sharp$:	$X \in [0;12] \wedge Y \in [0;8]$	(linear cost)
polyhedra $\mathcal{D}_p^\sharp$:	$6X + 11Y \geq 33 \wedge \cdots$	(exponential cost)

$\Longrightarrow$ trade-off cost vs. precision and expressiveness.

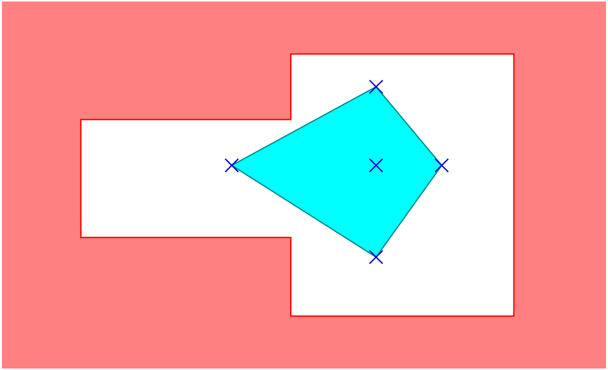
Correctness proofs and false alarms



Goal: prove that the program never enters an error state

The program is correct $(\text{blue} \cap \text{red} = \emptyset)$

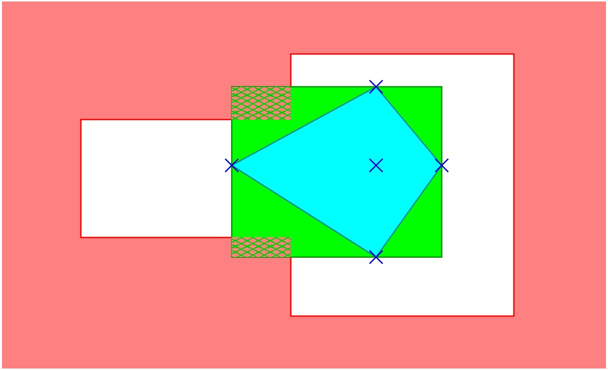
Correctness proofs and false alarms



Goal: prove that the program never enters an error state

The program is correct $(\mathrm{blue} \cap \mathrm{red} = \emptyset)$
The polyhedra domain can prove the correctness $(\mathrm{cyan} \cap \mathrm{red} = \emptyset)$

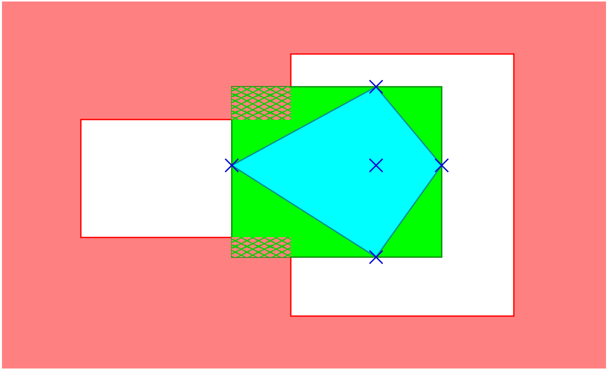
Correctness proofs and false alarms



Goal: prove that the program never enters an error state

The program is correct $(\text{blue} \cap \text{red} = \emptyset)$
The polyhedra domain can prove the correctness $(\text{cyan} \cap \text{red} = \emptyset)$
The intervals domain cannot $(\text{green} \cap \text{red} \neq \emptyset$, false alarm$)$

Correctness proofs and false alarms



Goal: prove that the program never enters an error state

The program is correct $(\mathrm{blue} \cap \mathrm{red} = \emptyset)$
The polyhedra domain can prove the correctness $(\mathrm{cyan} \cap \mathrm{red} = \emptyset)$
The intervals domain cannot $(\mathrm{green} \cap \mathrm{red} \neq \emptyset$, false alarm)

Trade-off between cost and precision (number of false alarms)

Overview

- **Rational domains**
  - concrete & abstract semantics of a toy language
  - interval domain
  - polyhedra domain
- **Floating-point domains**
  - linearization of float expressions
  - float polyhedra
- **Binary representation aware domains**
  - machine integers
  - memory abstraction
  - binary float domains
- **Application:** Astrée analyzer

Rational Domains

Toy language: syntax

arithmetic expressions:

`exp`	$::=$	`V`	variable `V` $\in \mathcal{V}$
	$\mid$	$-$`exp`	negation
	$\mid$	`exp` $\diamond$ `exp`	binary operation: $\diamond \in \{\,+, -, \times, /\,\}$
	$\mid$	$[c, c']$	constant range, $c, c' \in \mathbb{Q} \cup \{\,\pm\infty\,\}$
			(c is a shorthand for $[c, c]$)

programs:

`prog`	$::=$	`V := exp`	assignment
	$\mid$	`if exp` $\bowtie$ `0 then prog else prog fi`	test
	$\mid$	`while exp` $\bowtie$ `0 do prog done`	loop
	$\mid$	`prog; prog`	sequence

Finite set $\mathcal{V}$ of variables, with value in $\mathbb{Q}$

(later extended to floats $\mathbb{F}$ and machine integers $\mathbb{M}$)

Concrete semantics

Semantics of expressions: $\mathrm{E}[\![\, e \,]\!] : (\mathcal{V} \to \mathbb{Q}) \to \mathcal{P}(\mathbb{Q})$

The evaluation of e in ρ gives a set of values:

$$
\begin{array}{lcl}
\mathrm{E}[\![\, [c, c'] \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, x \in \mathbb{Q} \mid c \leq x \leq c' \,\} \\
\mathrm{E}[\![\, \mathtt{V} \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, \rho(\mathtt{V}) \,\} \\
\mathrm{E}[\![\, -e \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, -v \mid v \in \mathrm{E}[\![\, e \,]\!]\, \rho \,\} \\
\mathrm{E}[\![\, e_1 + e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, v_1 + v_2 \mid v_1 \in \mathrm{E}[\![\, e_1 \,]\!]\, \rho,\, v_2 \in \mathrm{E}[\![\, e_2 \,]\!]\, \rho \,\} \\
\mathrm{E}[\![\, e_1 - e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, v_1 - v_2 \mid v_1 \in \mathrm{E}[\![\, e_1 \,]\!]\, \rho,\, v_2 \in \mathrm{E}[\![\, e_2 \,]\!]\, \rho \,\} \\
\mathrm{E}[\![\, e_1 \times e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, v_1 \times v_2 \mid v_1 \in \mathrm{E}[\![\, e_1 \,]\!]\, \rho,\, v_2 \in \mathrm{E}[\![\, e_2 \,]\!]\, \rho \,\} \\
\mathrm{E}[\![\, e_1 / e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & \{\, v_1 / v_2 \mid v_1 \in \mathrm{E}[\![\, e_1 \,]\!]\, \rho,\, v_2 \in \mathrm{E}[\![\, e_2 \,]\!]\, \rho,\; v_2 \neq 0 \,\}
\end{array}
$$

Concrete semantics

Semantics of programs: $\mathrm{C}[\![\, p \,]\!] : \mathcal{D} \to \mathcal{D}$

where $\mathcal{D} \stackrel{\text{def}}{=} \mathcal{P}(\mathcal{V} \to \mathbb{Q})$

A transfer function for p defines a relation on environments $\rho \in \mathcal{D}$:

$$\mathrm{C}[\![\, \mathtt{V} := e \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} \{\, \rho[\, \mathtt{V} \mapsto v \,] \mid \rho \in \mathcal{X},\ v \in \mathrm{E}[\![\, e \,]\!]\,\rho \,\}$$

$$\mathrm{C}[\![\, e \bowtie 0 \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} \{\, \rho \mid \rho \in \mathcal{X},\ \exists v \in \mathrm{E}[\![\, e \,]\!]\,\rho,\ v \bowtie 0 \,\}$$

$$\mathrm{C}[\![\, b_1; b_2 \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} \mathrm{C}[\![\, b_2 \,]\!]\,(\mathrm{C}[\![\, b_1 \,]\!]\,\mathcal{X})$$

$$\mathrm{C}[\![\, \mathtt{if}\ e \bowtie 0\ \mathtt{then}\ b_1\ \mathtt{else}\ b_2 \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} (\mathrm{C}[\![\, b_1 \,]\!] \circ \mathrm{C}[\![\, e \bowtie 0 \,]\!])\mathcal{X} \cup (\mathrm{C}[\![\, b_2 \,]\!] \circ \mathrm{C}[\![\, e \not\bowtie 0 \,]\!])\mathcal{X}$$

$$\mathrm{C}[\![\, \mathtt{while}\ e \bowtie 0\ \mathtt{do}\ b\ \mathtt{done} \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} \mathrm{C}[\![\, e \not\bowtie 0 \,]\!]\,(\mathsf{lfp}\,\lambda\mathcal{Y}.\mathcal{X} \cup (\mathrm{C}[\![\, b \,]\!] \circ \mathrm{C}[\![\, e \bowtie 0 \,]\!])\mathcal{Y})$$

It relates the environments after the execution of a command to the environments before.

Abstract domains

- Abstract elements:
  - $\mathcal{D}^\sharp$ set of computer-representable elements
  - $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$ concretization
  - $\subseteq^\sharp$ approximation order: $\mathcal{X}^\sharp \subseteq^\sharp \mathcal{Y}^\sharp \implies \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathcal{Y}^\sharp)$
- Abstract operators:
  - $\mathrm{C}^\sharp[\![\, c \,]\!] : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ and $\cup^\sharp : (\mathcal{D}^\sharp \times \mathcal{D}^\sharp) \to \mathcal{D}^\sharp$
  - soundness: $(\mathrm{C}[\![\, c \,]\!] \circ \gamma)(\mathcal{X}^\sharp) \subseteq (\gamma \circ \mathrm{C}^\sharp[\![\, c \,]\!])(\mathcal{X}^\sharp)$
    $\gamma(\mathcal{X}^\sharp) \cup \gamma(\mathcal{Y}^\sharp) \subseteq \gamma(\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp)$
- Fixpoint extrapolation
  - $\triangledown : (\mathcal{D}^\sharp \times \mathcal{D}^\sharp) \to \mathcal{D}^\sharp$ widening
  - soundness: $\gamma(\mathcal{X}^\sharp) \cup \gamma(\mathcal{Y}^\sharp) \subseteq \gamma(\mathcal{X}^\sharp \triangledown \mathcal{Y}^\sharp)$
  - termination: $\forall$ sequence $(\mathcal{Y}_i^\sharp)_{i \in \mathbb{N}}$
    the sequence $\mathcal{X}_0^\sharp = \mathcal{Y}_0^\sharp,\ \mathcal{X}_{i+1}^\sharp = \mathcal{X}_i^\sharp \triangledown \mathcal{Y}_{i+1}^\sharp$
    stabilizes in finite time: $\exists n < \omega,\ \mathcal{X}_{n+1}^\sharp = \mathcal{X}_n^\sharp$

Both semantics and algorithmic aspects.

Galois connection

Galois connection definition: $(\mathcal{D}, \subseteq) \xleftrightarrow[\alpha]{\gamma} (\mathcal{D}^\sharp, \subseteq^\sharp)$

- monotonic concretization $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$
- monotonic abstraction $\alpha : \mathcal{D} \to \mathcal{D}^\sharp$
- $\forall \mathcal{X} \in \mathcal{D}: \forall \mathcal{Y}^\sharp \in \mathcal{D}^\sharp: \alpha(\mathcal{X}) \subseteq^\sharp \mathcal{Y}^\sharp \iff \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp)$

Application: optimal abstractions

- elements $\mathcal{X} \in \mathcal{D}$ have a best abstraction: $\alpha(\mathcal{X})$
  $\alpha(\mathcal{X}) = \bigcap^\sharp \{ \mathcal{Y}^\sharp \mid \mathcal{X} \subseteq \gamma(\mathcal{Y}^\sharp) \}$
- functions $F : \mathcal{D} \to \mathcal{D}$ have a best abstraction:
  $F^\sharp \stackrel{\text{def}}{=} \alpha \circ F \circ \gamma$
- however optimality does not compose
  $\alpha \circ (F_1 \circ F_2) \circ \gamma \subsetneq (\alpha \circ F_1 \circ \gamma) \circ (\alpha \circ F_2 \circ \gamma) \qquad (\gamma \circ \alpha \supsetneq id)$
- provides semantic aspects only, no algorithm!

Abstract semantics

Given by the abstract domain:

- sound $\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]$, $\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]$, $\cup^\sharp$
- sound and terminating $\triangledown$

Derived analysis: from the concrete...

$$\mathrm{C}[\![\, b_1 ; b_2 \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=} \mathrm{C}[\![\, b_2 \,]\!]\,(\mathrm{C}[\![\, b_1 \,]\!]\,\mathcal{X})$$

$$\mathrm{C}[\![\, \mathtt{if}\ e \bowtie 0\ \mathtt{then}\ b_1\ \mathtt{else}\ b_2 \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=}$$
$$(\mathrm{C}[\![\, b_1 \,]\!] \circ \mathrm{C}[\![\, e \bowtie 0 \,]\!])\mathcal{X} \cup (\mathrm{C}[\![\, b_2 \,]\!] \circ \mathrm{C}[\![\, e \not\bowtie 0 \,]\!])\mathcal{X}$$

$$\mathrm{C}[\![\, \mathtt{while}\ e \bowtie 0\ \mathtt{do}\ b\ \mathtt{done} \,]\!]\,\mathcal{X} \stackrel{\text{def}}{=}$$
$$\mathrm{C}[\![\, e \not\bowtie 0 \,]\!]\,(\mathrm{lfp}\,\lambda \mathcal{Y}.\mathcal{X} \cup (\mathrm{C}[\![\, b \,]\!] \circ \mathrm{C}[\![\, e \bowtie 0 \,]\!])\mathcal{Y})$$

Abstract semantics

Given by the abstract domain:

- sound $\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]$, $\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]$, $\cup^\sharp$
- sound and terminating $\triangledown$

Derived analysis: …to the abstract

$$\mathrm{C}^\sharp[\![\, b_1; b_2 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathrm{C}^\sharp[\![\, b_2 \,]\!]\, (\mathrm{C}^\sharp[\![\, b_1 \,]\!]\, \mathcal{X}^\sharp)$$

$$\mathrm{C}^\sharp[\![\, \mathtt{if}\ e \bowtie 0\ \mathtt{then}\ b_1\ \mathtt{else}\ b_2 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=}$$
$$(\mathrm{C}^\sharp[\![\, b_1 \,]\!] \circ \mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!])\mathcal{X}^\sharp \cup^\sharp (\mathrm{C}^\sharp[\![\, b_2 \,]\!] \circ \mathrm{C}^\sharp[\![\, e \not\bowtie 0 \,]\!])\mathcal{X}^\sharp$$

$$\mathrm{C}^\sharp[\![\, \mathtt{while}\ e \bowtie 0\ \mathtt{do}\ b\ \mathtt{done} \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=}$$
$$\mathrm{C}^\sharp[\![\, e \not\bowtie 0 \,]\!]\, (\lim \lambda \mathcal{Y}^\sharp . \mathcal{Y}^\sharp \,\triangledown\, (\mathcal{X}^\sharp \cup^\sharp (\mathrm{C}^\sharp[\![\, b \,]\!] \circ \mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!])\mathcal{Y}^\sharp))$$

The derived analysis is sound and terminates.

Intervals domain

Intervals lattice

$\mathcal{B}^\sharp \stackrel{\text{def}}{=} \{\, [a,b] \mid a \in \mathbb{Q} \cup \{\, -\infty \,\},\ b \in \mathbb{Q} \cup \{\, +\infty \,\},\ a \leq b \,\}$

[Cousot 76]

Galois connection: $\mathcal{P}(\mathbb{Q}) \underset{\alpha_b}{\overset{\gamma_b}{\leftrightarrows}} \mathcal{B}^\sharp \cup \{\, \bot^\sharp \,\}$

$$
\begin{aligned}
\gamma([a,b]) &\stackrel{\text{def}}{=} \{\, x \in \mathbb{Q} \mid a \leq x \leq b \,\} \\
\alpha(\mathcal{X}) &\stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{X} = \emptyset \\ [\min \mathcal{X}, \max \mathcal{X}] & \text{otherwise} \end{cases}
\end{aligned}
$$

(α is not always defined, but $\alpha \circ F \circ \gamma$ is generally defined)

Partial order:

$$
\begin{aligned}
[a,b] \subseteq^\sharp [c,d] &\stackrel{\text{def}}{\iff} a \geq c \text{ and } b \leq d \\
\top^\sharp &\stackrel{\text{def}}{=}]-\infty, +\infty[\\
[a,b] \cup^\sharp [c,d] &\stackrel{\text{def}}{=} [\min(a,c), \max(b,d)] \\
[a,b] \cap^\sharp [c,d] &\stackrel{\text{def}}{=} \begin{cases} [\max(a,c), \min(b,d)] & \text{if } \max \leq \min \\ \bot^\sharp & \text{otherwise} \end{cases}
\end{aligned}
$$

Derived abstract domain

Pointwise lifting to an abstraction of $\mathcal{P}(\mathcal{V} \to \mathbb{Q})$:

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} (\mathcal{V} \to \mathcal{B}^\sharp) \cup \{\, \bot^\sharp \,\}$$

$$\top^\sharp \stackrel{\text{def}}{=} \lambda \mathtt{V}.\top^\sharp$$

$$\mathcal{X}^\sharp \subseteq^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{\iff} \mathcal{X}^\sharp = \bot^\sharp \vee (\mathcal{X}^\sharp, \mathcal{Y}^\sharp \neq \bot^\sharp \wedge \forall \mathtt{V}: \mathcal{X}^\sharp(\mathtt{V}) \subseteq^\sharp \mathcal{Y}^\sharp(\mathtt{V}))$$

$$\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \begin{cases} \mathcal{Y}^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \\ \mathcal{X}^\sharp & \text{if } \mathcal{Y}^\sharp = \bot^\sharp \\ \lambda \mathtt{V}.\mathcal{X}^\sharp(\mathtt{V}) \cup^\sharp \mathcal{Y}^\sharp(\mathtt{V}) & \text{otherwise} \end{cases}$$

$$\mathcal{X}^\sharp \cap^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{X}^\sharp = \bot^\sharp \text{ or } \mathcal{Y}^\sharp = \bot^\sharp \\ \bot^\sharp & \text{if } \exists \mathtt{V}: \mathcal{X}^\sharp(\mathtt{V}) \cap^\sharp \mathcal{Y}^\sharp(\mathtt{V}) = \bot^\sharp \\ \lambda \mathtt{V}.\mathcal{X}^\sharp(\mathtt{V}) \cap^\sharp \mathcal{Y}^\sharp(\mathtt{V}) & \text{otherwise} \end{cases}$$

Interval abstract arithmetic operators

Based on interval arithmetic [Moore 66]

$$
\begin{array}{rcl}
[c, c']^\sharp & \stackrel{\text{def}}{=} & [c, c'] \\
-^\sharp\,[a, b] & \stackrel{\text{def}}{=} & [-b, -a] \\
[a, b] +^\sharp [c, d] & \stackrel{\text{def}}{=} & [a + c, b + d] \\
[a, b] -^\sharp [c, d] & \stackrel{\text{def}}{=} & [a - d, b - c] \\
[a, b] \times^\sharp [c, d] & \stackrel{\text{def}}{=} & [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\
[a, b] /^\sharp [c, d] & \stackrel{\text{def}}{=} & \cdots
\end{array}
$$

where $\pm\infty \times 0 = 0$.

Interval abstract assignment

Abstract evaluation of expressions: $\mathrm{E}^\sharp[\![\, e \,]\!] : \mathcal{D}^\sharp \to \mathcal{B}^\sharp$

$$\mathrm{E}^\sharp[\![\, e \,]\!]\, \bot^\sharp \stackrel{\text{def}}{=} \bot^\sharp$$

if $\mathcal{X}^\sharp \neq \bot^\sharp$:

$$\begin{aligned}
\mathrm{E}^\sharp[\![\, [c, c'] \,]\!]\, \mathcal{X}^\sharp &\stackrel{\text{def}}{=} [c, c']^\sharp \\
\mathrm{E}^\sharp[\![\, \mathtt{V} \,]\!]\, \mathcal{X}^\sharp &\stackrel{\text{def}}{=} \mathcal{X}^\sharp(\mathtt{V}) \\
\mathrm{E}^\sharp[\![\, -e \,]\!]\, \mathcal{X}^\sharp &\stackrel{\text{def}}{=} -^\sharp\, \mathrm{E}^\sharp[\![\, e \,]\!]\, \mathcal{X}^\sharp \\
\mathrm{E}^\sharp[\![\, e_1 \diamond e_2 \,]\!]\, \mathcal{X}^\sharp &\stackrel{\text{def}}{=} \mathrm{E}^\sharp[\![\, e_1 \,]\!]\, \mathcal{X}^\sharp \diamond^\sharp \mathrm{E}^\sharp[\![\, e_2 \,]\!]\, \mathcal{X}^\sharp
\end{aligned}$$

Abstract assignment:

$$\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } \mathcal{V}^\sharp = \bot^\sharp \\ \mathcal{X}^\sharp[\mathtt{V} \mapsto \mathcal{V}^\sharp] & \text{otherwise} \end{cases}$$

where $\mathcal{V}^\sharp = \mathrm{E}^\sharp[\![\, e \,]\!]\, \mathcal{X}^\sharp$.

Note: $\mathrm{C}^\sharp[\![\, V := e \,]\!]$ may not be optimal, even though each $\diamond^\sharp$ is.

Interval abstract tests

If $\mathcal{X}^\sharp(\mathtt{X}) = [a, b]$ and $\mathcal{X}^\sharp(\mathtt{Y}) = [c, d]$, we can define:

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - c \leq \mathtt{0} \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } a > c \\ \mathcal{X}^\sharp[\, \mathtt{X} \mapsto [a, \min(b, c)]\,] & \text{otherwise} \end{cases}$$

$$\mathrm{C}^\sharp[\![\, \mathtt{X} - \mathtt{Y} \leq \mathtt{0} \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \begin{cases} \bot^\sharp & \text{if } a > d \\ \mathcal{X}^\sharp[\; \mathtt{X} \mapsto [a, \min(b, d)], & \text{otherwise} \\ \qquad \mathtt{Y} \mapsto [\max(c, a), d]\,] & \end{cases}$$

General case: constraint programming (HC4)

Note: fall-back operators

- $\mathrm{C}^\sharp[\![\, e \bowtie 0 \,]\!]\, \mathcal{X}^\sharp = \mathcal{X}^\sharp$ is always sound
- $\mathrm{C}^\sharp[\![\, \mathtt{X} := e \,]\!]\, \mathcal{X}^\sharp = \mathcal{X}^\sharp[\mathtt{X} \mapsto \top^\sharp]$ is always sound

Interval widening

Widening on non-relational domains:

Given a value widening $\triangledown\colon \mathcal{B}^\sharp \times \mathcal{B}^\sharp \to \mathcal{B}^\sharp$,
we extend it point-wisely into a widening $\triangledown\colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$:

$$\mathcal{X}^\sharp \triangledown \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \lambda \mathtt{V}.\mathcal{X}^\sharp(\mathtt{V}) \triangledown \mathcal{Y}^\sharp(\mathtt{V})$$

Interval widening example:

$$\begin{array}{rclcl}
\bot^\sharp & \triangledown & X^\sharp & \stackrel{\text{def}}{=} & X^\sharp \\
{[a,b]} & \triangledown & [c,d] & \stackrel{\text{def}}{=} & \left[\begin{cases} a & \text{if } a \le c \\ -\infty & \text{otherwise} \end{cases}, \begin{cases} b & \text{if } b \ge d \\ +\infty & \text{otherwise} \end{cases} \right]
\end{array}$$

Unstable bounds are set to $\pm\infty$

Analysis with widening example

```
X:=0;
while • X<40 do
  X:=X+3
done
```

Analysis with widening example

```
X:=0;
while • X<40 do
  X:=X+3
done
```

We must compute:

$\mathrm{C}^\sharp[\![\, \mathtt{X} \geq 40 \,]\!]\, (\lim \lambda \mathcal{Y}^\sharp . \mathcal{Y}^\sharp \,\triangledown\, (\mathcal{X}^\sharp \cup^\sharp \mathrm{C}^\sharp[\![\, \mathtt{X} := \mathtt{X}+3 \,]\!]\, (\mathrm{C}^\sharp[\![\, \mathtt{X} < 40 \,]\!]\, \mathcal{Y}^\sharp)))$

- $\mathcal{Y}_0^\sharp = \mathcal{X}^\sharp = [0, 0]$
- $\mathcal{Y}_1^\sharp = \mathcal{Y}_0^\sharp \,\triangledown\, (\mathcal{X}^\sharp \cup^\sharp (\mathcal{Y}_0^\sharp +^\sharp [3, 3])) = [0, 0] \,\triangledown\, ([0, 0] \cup^\sharp [3, 3]) = [0, +\infty]$
- $\mathcal{Y}_2^\sharp = \mathcal{Y}_1^\sharp \,\triangledown\, (\mathcal{X}^\sharp \cup^\sharp (\mathcal{Y}_1^\sharp +^\sharp [3, 3])) = [0, +\infty] \,\triangledown\, ([0, 0] \cup^\sharp [3, 42]) = \mathcal{Y}_1^\sharp$
- $\mathrm{C}^\sharp[\![\, \mathtt{X} \geq 40 \,]\!]\, (\mathcal{Y}_2^\sharp) = [42, +\infty]$

Decreasing iterations: to improve the precision

- after stabilization, continue iterating without $\triangledown$ (use $\cap$)
- in our case, $\mathcal{Y}_3^\sharp = [0, 42]$, so $\mathrm{C}^\sharp[\![\, \mathtt{X} \geq 40 \,]\!]\, (\mathcal{Y}_3^\sharp) = [40, 42]$

Polyhedra Domain

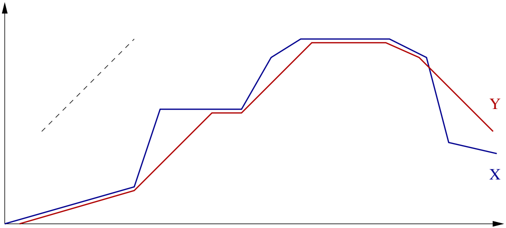
The need for relational domains

Non-relation domains cannot represent variable relationships

Rate limiter

```
Y:=0; while • true do
  X:=[-128,128]; D:=[0,16];
  S:=Y; Y:=X; R:=X-S;
  if R<=-D then Y:=S-D fi;
  if R>=D then Y:=S+D fi
done
```

X: input signal
Y: output signal
S: last output
R: delta Y-S
D: max. allowed for |R|



The need for relational domains

Non-relation domains cannot represent variable relationships

Rate limiter

```
Y:=0; while • true do
  X:=[-128,128]; D:=[0,16];
  S:=Y; Y:=X; R:=X-S;
  if R<=-D then Y:=S-D fi;
  if R>=D then Y:=S+D fi
done
```

X:	input signal
Y:	output signal
S:	last output
R:	delta Y-S
D:	max. allowed for $|R|$

Iterations in the interval domain (without widening):

$\mathcal{X}^{\sharp 0}_{\bullet}$	$\mathcal{X}^{\sharp 1}_{\bullet}$	$\mathcal{X}^{\sharp 2}_{\bullet}$	...	$\mathcal{X}^{\sharp n}_{\bullet}$
$Y = 0$	$|Y| \leq 144$	$|Y| \leq 160$	...	$|Y| \leq 128 + 16n$

In fact, $Y \in [-128, 128]$ always holds.

To prove that, e.g. $Y \geq -128$, we must be able to:

- represent the properties $R = X - S$ and $R \leq -D$
- combine them to deduce $S - X \geq D$, and then $Y = S - D \geq X$

Polyhedra domain

Domain proposed by [Cousot Halbwachs 78]
to infer conjunctions of affine inequalities $\bigwedge_j \left(\sum_{i=1}^{n} \alpha_{ij} \mathtt{V}_i \geq \beta_j\right)$.

Abstract elements:

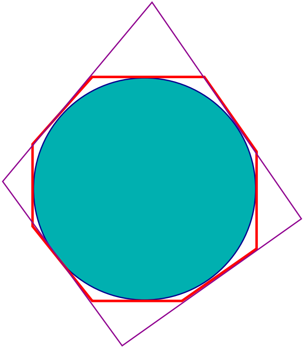
- $\mathit{LinCons} \stackrel{\text{def}}{=}$ affine constraints over $\mathcal{V}$ with coefficients in $\mathbb{Q}$
- $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{P}_{\text{finite}}(\mathit{LinCons})$

Concretization:

$$\gamma(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} \{\, \rho \in \mathcal{V} \to \mathbb{Q} \mid \forall c \in \mathcal{X}^\sharp,\, \rho \models c \,\}$$

- $\gamma(\mathcal{X}^\sharp)$ is a closed convex polyhedron of $(\mathcal{V} \to \mathbb{Q}) \simeq \mathbb{Q}^{|\mathcal{V}|}$
- $\gamma(\mathcal{X}^\sharp)$ may be empty, bounded, or unbounded
- γ is not injective

Polyhedra representations



- No memory bound on the representations (even minimal ones)
- No best abstraction α
- Dual representation using generators
  (double description method)

Polyhedra algorithms

Fourier-Motzkin elimination:

Fourier$(\mathcal{X}^\sharp, \mathtt{V}_k)$ eliminates $\mathtt{V}_k$ from all the constraints in $\mathcal{X}^\sharp$:

$$
\begin{array}{l}
\textit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_k) \stackrel{\text{def}}{=} \\
\quad \{\ (\sum_i \alpha_i \mathtt{V}_i \geq \beta) \in \mathcal{X}^\sharp \mid \alpha_k = 0\ \} \cup \\
\quad \{\ (-\alpha_k^-)c^+ + \alpha_k^+ c^- \mid \begin{array}{l} c^+ = (\sum_i \alpha_i^+ \mathtt{V}_i \geq \beta^+) \in \mathcal{X}^\sharp,\ \alpha_k^+ > 0, \\ c^- = (\sum_i \alpha_i^- \mathtt{V}_i \geq \beta^-) \in \mathcal{X}^\sharp,\ \alpha_k^- < 0\ \} \end{array}
\end{array}
$$

Semantics

$$\gamma(\textit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_k)) = \{\ \rho[\mathtt{V}_k \mapsto v] \mid v \in \mathbb{Q},\ \rho \in \gamma(\mathcal{X}^\sharp)\ \}$$

i.e., forget the value of $\mathtt{V}_k$

Polyhedra algorithms

Linear programming:

$$\textit{simplex}(\mathcal{X}^\sharp, \vec{\alpha}) \stackrel{\text{def}}{=} \min \{ \textstyle\sum_i \alpha_i \rho(\mathtt{V}_i) \mid \rho \in \gamma(\mathcal{X}^\sharp) \}$$

Application: remove redundant constraints:

for each $c = (\sum_i \alpha_i \mathtt{V}_i \geq \beta) \in \mathcal{X}^\sharp$
if $\beta \leq \textit{simplex}(\mathcal{X}^\sharp \setminus \{c\}, \vec{\alpha})$, then remove c from $\mathcal{X}^\sharp$

(e.g., *Fourier* causes a quadratic growth in constraint number, most of which are redundant)

Note: calling *simplex* many times can be costly

- use fast syntactic checks first
- check against the bounding-box first
- use *simplex* as a last resort

Polyhedra abstract operators

Order: $\subseteq^\sharp$

$$\mathcal{X}^\sharp \subseteq^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{\iff} \forall (\textstyle\sum_i \alpha_i \mathtt{V}_i \geq \beta) \in \mathcal{Y}^\sharp,\ \mathit{simplex}(\mathcal{X}^\sharp, \vec{\alpha}) \geq \beta$$

$$\stackrel{\text{def}}{\iff} \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathcal{Y}^\sharp)$$

$$\mathcal{X}^\sharp =^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{\iff} \mathcal{X}^\sharp \subseteq^\sharp \mathcal{Y}^\sharp \wedge \mathcal{Y}^\sharp \subseteq^\sharp \mathcal{X}^\sharp$$

Polyhedra abstract operators (cont.)

Convex hull:

- Express a point $\vec{V} \in \mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp$ as a convex combination:
  $\vec{V} = \sigma\vec{X} + \sigma'\vec{Y}$ for $\vec{X} \in \mathcal{X}^\sharp$, $\vec{Y} \in \mathcal{Y}^\sharp$, $\sigma + \sigma' = 1$, $\sigma, \sigma' \geq 0$
- as $\sigma\vec{X} + \sigma'\vec{Y}$ is quadratic
  we consider instead: $\vec{V} = \vec{X} + \vec{Y}$ with $\vec{X}/\sigma \in \mathcal{X}^\sharp$, $\vec{Y}/\sigma' \in \mathcal{Y}^\sharp$
  i.e., $\vec{X} \in \sigma\mathcal{X}^\sharp$, $\vec{Y} \in \sigma'\mathcal{Y}^\sharp$
  (adds closure points on unbounded polyhedra)

Formally:

$$\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{=}$$

$$\begin{array}{ll} \textit{Fourier}(& \{\, (\sum_j \alpha_j \mathtt{X}_j - \beta\sigma \geq 0) \mid (\sum_j \alpha_j \mathtt{V}_j \geq \beta) \in \mathcal{X}^\sharp \,\} \;\; \cup \\ & \{\, (\sum_j \alpha_j \mathtt{Y}_j - \beta\sigma' \geq 0) \mid (\sum_j \alpha_j \mathtt{V}_j \geq \beta) \in \mathcal{Y}^\sharp \,\} \;\; \cup \\ & \{\, \mathtt{V}_j = \mathtt{X}_j + \mathtt{Y}_j \mid \mathtt{V}_j \in \mathcal{V} \,\} \cup \{\, \sigma \geq 0,\ \sigma' \geq 0,\ \sigma + \sigma' = 1 \,\}, \\ & \{\, \mathtt{X}_j, \mathtt{Y}_j \mid \mathtt{V}_j \in \mathcal{V} \,\} \;\cup\; \{\, \sigma, \sigma' \,\}\,) \end{array}$$

[Benoi et al. 96]

Polyhedra abstract operators (cont.)

Precise abstract commands: (exact)

$$\mathrm{C}^\sharp[\![\, \sum_i \alpha_i \mathtt{V}_i + \beta \le 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathcal{X}^\sharp \cup \{(\sum_i \alpha_i \mathtt{V}_i + \beta \le 0)\}$$

$$\mathrm{C}^\sharp[\![\, \mathtt{V}_j := [-\infty, +\infty] \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_j))$$

$$\mathrm{C}^\sharp[\![\, \mathtt{V}_j := \sum_i \alpha_i \mathtt{V}_i + \beta^\sharp \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=}$$
$$\mathit{subst}(\mathtt{V} \mapsto \mathtt{V}_i,\, \mathit{Fourier}((\mathcal{X}^\sharp \cup \{\mathtt{V} = \sum_i \alpha_i \mathtt{V}_i + \beta\}),\, \mathtt{V}_j))$$

Fallback abstract commands: (coarse but sound)

$$\mathrm{C}^\sharp[\![\, e \le 0 \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathcal{X}^\sharp$$

$$\mathrm{C}^\sharp[\![\, \mathtt{V}_j := e \,]\!]\, \mathcal{X}^\sharp \stackrel{\text{def}}{=} \mathit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_j)$$

alternate solution:
apply interval abstract commands to the bounding box

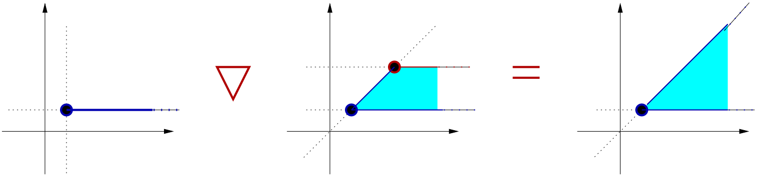
Polyhedra widening

Classic widening $\triangledown$ in $\mathcal{D}^\sharp$

$$\mathcal{X}^\sharp \triangledown \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \{\, c \in \mathcal{X}^\sharp \mid \mathcal{Y}^\sharp \subseteq^\sharp \{c\} \,\} \;\cup\; \{\, c \in \mathcal{Y}^\sharp \mid \exists c' \in \mathcal{X}^\sharp,\ \mathcal{X}^\sharp =^\sharp (\mathcal{X}^\sharp \setminus c') \cup \{c\} \,\}$$

- suppress unstable constraints $c \in \mathcal{X}^\sharp$, $\mathcal{Y}^\sharp \not\subseteq^\sharp \{c\}$
- add back constraints $c \in \mathcal{Y}^\sharp$ equivalent to those in $\mathcal{X}^\sharp$ i.e., when $\exists c' \in \mathcal{X}^\sharp,\ \mathcal{X}^\sharp =^\sharp (\mathcal{X}^\sharp \setminus c') \cup \{c\}$. ($\mathcal{X}^\sharp$ and $\mathcal{Y}^\sharp$ must have no redundant constraint)

Example:



Floating-point domains

Floating-point uses

Two independent problems:

- **Implement the analyzer using floating-point**

  goal: trade precision for efficiency

  exact rational arithmetic can be costly
  coefficients can grow large (polyhedra)

- **Analyze floating-point programs**

  goal: catch run-time errors caused by rounding
  (overflow, division by 0, . . .)

Also: a floating-point analyzer for floating-point programs.

Challenge: how to stay **sound**?

Floating-point computations

The set of floating-point numbers is not closed under $+$, $-$, $\times$, $/$:

- every result is rounded to a representable float,
- an overflow or division by 0 generates $+\infty$ or $-\infty$ (overflow);
- small numbers are truncated to $+0$ or -0 (underflow);
- some operations are invalid ($0/0$, $(+\infty) + (-\infty)$, etc.) and return *NaN*.

Observable semantics:

- overflows and *NaNs* halt the program with an error $\mathcal{O}$,
- rounding and underflow are not errors,
- we do not distinguish between $+0$ and -0.

$\Longrightarrow$ variable values live in a finite subset $\mathbb{F}$ of $\mathbb{Q}$,
expression values live in $\mathbb{F} \cup \{\mathcal{O}\}$.

Floating-point expressions

Floating-point expressions $\mathtt{exp}_f$

$$
\begin{array}{lcll}
\mathtt{exp}_f & ::= & [c, c'] & \text{constant range } c, c' \in \mathbb{F},\ c \le c' \\
 & | & \mathtt{V} & \text{variable } \mathtt{V} \in \mathcal{V} \\
 & | & \ominus\ \mathtt{exp}_f & \text{negation} \\
 & | & \mathtt{exp}_f \odot_r \mathtt{exp}_f & \text{operator } \odot \in \{\oplus, \ominus, \otimes, \oslash\}
\end{array}
$$

(we use circled operators to distinguish them from operators in $\mathbb{Q}$)

Concrete semantics of expressions

Semantics of rounding: $R_r \colon \mathbb{Q} \to \mathbb{F} \cup \{\mathcal{O}\}$.

Example definition:

$$R_{+\infty}(x) \stackrel{\text{def}}{=} \begin{cases} \min \{\, y \in \mathbb{F} \mid y \geq x \,\} & \text{if } x \leq Mf \\ \mathcal{O} & \text{if } x > Mf \end{cases}$$

$$R_{-\infty}(x) \stackrel{\text{def}}{=} \begin{cases} \max \{\, y \in \mathbb{F} \mid y \leq x \,\} & \text{if } x \geq -Mf \\ \mathcal{O} & \text{if } x < -Mf \end{cases}$$

Notes:

- $\forall x, r,\ R_{-\infty}(x) \leq R_r(x) \leq R_{+\infty}(x)$
- $\forall r, R_r$ is monotonic

Concrete semantics of expressions (cont.)

$\mathrm{E}[\![\, e_f \,]\!] : (\mathcal{V} \to \mathbb{F}) \to \mathcal{P}(\mathbb{F} \cup \{\mathcal{O}\})$ (expression semantics)

$$
\begin{array}{lcl}
\mathrm{E}[\![\, \mathtt{V} \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\ \rho(\mathtt{V})\ \} \\
\mathrm{E}[\![\, [c, c'] \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\ x \in \mathbb{F} \mid c \le x \le c'\ \} \\
\mathrm{E}[\![\, \ominus\ e_f \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\ -x \mid x \in \mathrm{E}[\![\, e_f \,]\!]\,\rho \cap \mathbb{F}\ \} \cup (\{\mathcal{O}\} \cap \mathrm{E}[\![\, e_f \,]\!]\,\rho) \\
\mathrm{E}[\![\, e_f \odot_r e'_f \,]\!]\,\rho & \stackrel{\text{def}}{=} &
\end{array}
$$

$$
\begin{array}{l}
\{\ R_r(x \cdot y) \mid x \in \mathrm{E}[\![\, e_f \,]\!]\,\rho \cap \mathbb{F},\ y \in \mathrm{E}[\![\, e'_f \,]\!]\,\rho \cap \mathbb{F}\ \} \cup \\
\{\ \mathcal{O} \mid\ \text{if } \mathcal{O} \in \mathrm{E}[\![\, e_f \,]\!]\,\rho \cup \mathrm{E}[\![\, e'_f \,]\!]\,\rho\ \} \\
\{\ \mathcal{O} \mid\ \text{if } 0 \in \mathrm{E}[\![\, e'_f \,]\!]\,\rho \text{ and } \odot = \oslash\ \}
\end{array}
$$

$\mathrm{C}[\![\, c \,]\!] : \mathcal{P}(\mathcal{V} \to \mathbb{F}) \to \mathcal{P}((\mathcal{V} \to \mathbb{F}) \cup \{\mathcal{O}\})$ (command semantics)

$$
\begin{array}{lcl}
\mathrm{C}[\![\, \mathtt{X} := e_f \,]\!]\,\mathcal{X} & \stackrel{\text{def}}{=} & \{\ \rho[\,\mathtt{X} \mapsto v\,] \mid \rho \in \mathcal{X},\ v \in \mathrm{E}[\![\, e_f \,]\!]\,\rho \cap \mathbb{F}\ \} \\
& & \cup\ (\{\mathcal{O}\} \cap \mathrm{E}[\![\, e_f \,]\!]\,\mathcal{X}) \\
\mathrm{C}[\![\, e_f \le 0 \,]\!]\,\mathcal{X} & \stackrel{\text{def}}{=} & \{\ \rho \mid \rho \in \mathcal{X},\ \exists v \in \mathrm{E}[\![\, e_f \,]\!]\,\rho \cap \mathbb{F},\ v \le 0\ \} \\
& & \cup\ (\{\mathcal{O}\} \cap \mathrm{E}[\![\, e_f \,]\!]\,\mathcal{X})
\end{array}
$$

Floating-point interval domain

Representation: $\mathcal{B}^\sharp \stackrel{\text{def}}{=} \{\, [a,b] \mid a \in \mathbb{F},\ b \in \mathbb{F},\ a \le b \,\}$

Expression semantics: $\mathrm{E}^\sharp[\![\, \mathrm{exp}_f \,]\!] : (\mathcal{V} \to \mathcal{B}^\sharp) \to (\mathcal{B}^\sharp \cup \{\, \mathcal{O} \,\})$

$$
\begin{array}{lcl}
[a,b] \oplus^\sharp [a',b'] & \stackrel{\text{def}}{=} & [R_{-\infty}(a+a'), R_{+\infty}(b+b')] \\
[a,b] \ominus^\sharp [a',b'] & \stackrel{\text{def}}{=} & [R_{-\infty}(a-b'), R_{+\infty}(b-a')] \\
[a,b] \otimes^\sharp [a',b'] & \stackrel{\text{def}}{=} & [\ R_{-\infty}(\min(aa',ab',ba',bb')), \\
 & & \ \ R_{+\infty}(\max(aa',ab',ba',bb'))]
\end{array}
$$

- We suppose r is unknown and assume a worst case rounding.
- Soundness stems from the monotonicity of $R_{-\infty}$ and $R_{+\infty}$.
- Abstract operators also use float arithmetic (efficiency).

Error management

If some bound in $\mathrm{E}^\sharp[\![\, \mathrm{exp}_f \,]\!]$ evaluates to $\mathcal{O}$, we

- report the error to the user, and
- continue the evaluation with $\top^\sharp$.

Expression linearization

Floating-point issues in relational domains

Relational domains assume many powerful properties on $\mathbb{Q}$: associativity, distributivity,... that are not true on $\mathbb{F}$!

Example: Fourier-Motzkin elimination

$$\mathtt{X} - \mathtt{Y} \le c \quad \wedge \quad \mathtt{Y} - \mathtt{Z} \le d \quad \Longrightarrow \quad \mathtt{X} - \mathtt{Z} \le c + d$$

$$\mathtt{X} \ominus_n \mathtt{Y} \le c \quad \wedge \quad \mathtt{Y} \ominus_n \mathtt{Z} \le d \quad \not\Longrightarrow \quad \mathtt{X} \ominus_n \mathtt{Z} \le c \oplus_n d$$

$$(\mathtt{X} = 1,\ \mathtt{Y} = 10^{38},\ \mathtt{Z} = -1,\ c = \mathtt{X} \ominus_n \mathtt{Y} = -10^{38},$$
$$d = \mathtt{Y} \ominus_n \mathtt{Z} = 10^{38},\ c \oplus_n d = 0,\ \mathtt{X} \ominus_n \mathtt{Z} = 2 > 0)$$

We cannot manipulate float expressions as easily as rational ones!

Solution:

keep representing and manipulating rational expressions

- abstract float expressions from programs into rational ones
- feed them to a rational abstract domain
- (optional) implement the rational domain using floats

Affine interval forms

We put expressions in affine interval form: [Miné 04]

$$\mathtt{exp}_\ell ::= [a_0, b_0] + \sum_k [a_k, b_k] \times \mathtt{V}_k$$

Semantics:

$$\mathrm{E}[\![\, e_\ell \,]\!]\, \rho \stackrel{\text{def}}{=} \{\, c_0 + \sum_k c_k \times \rho(\mathtt{V}_k) \mid \forall i,\ c_i \in [a_i, b_i] \,\}$$

(evaluated in $\mathbb{Q}$)

Advantages:

- affine expressions are easy to manipulate
- interval coefficients allow non-determinism in expressions, hence, the opportunity for abstraction
- intervals can easily model rounding errors
- easy to design algorithms for $\mathrm{C}^\sharp[\![\, \mathtt{X} := e_\ell \,]\!]$ and $\mathrm{C}^\sharp[\![\, e_\ell \leq 0 \,]\!]$ in most domains

Affine interval form algebra

Operations on affine interval forms:

- adding $\boxplus$ and subtracting $\boxminus$ two forms
- multiplying $\boxtimes$ and dividing $\boxslash$ a form by an interval

Using interval arithmetic $\oplus^\sharp$, $\ominus^\sharp$, $\otimes^\sharp$, $\oslash^\sharp$:

$$(i_0 + \sum_k i_k \times \mathtt{V}_k) \boxplus (i'_0 + \sum_k i'_k \times \mathtt{V}_k) \stackrel{\text{def}}{=} (i_0 \oplus^\sharp i'_0) + \sum_k (i_k \oplus^\sharp i'_k) \times \mathtt{V}_k$$

$$i \boxtimes (i_0 + \sum_k i_k \times \mathtt{V}_k) \stackrel{\text{def}}{=} (i \otimes^\sharp i_0) + \sum_k (i \otimes^\sharp i_k) \times \mathtt{V}_k$$

...

Projection: $\pi_k : \mathcal{D}^\sharp \to \mathtt{exp}_\ell$

We suppose we are given an abstract interval projection operator π_k such that:

$$\pi_k(\mathcal{X}^\sharp) = [a, b] \text{ such that } [a, b] \supseteq \{\, \rho(\mathtt{V}_k) \mid \rho \in \gamma(\mathcal{X}^\sharp) \,\}.$$

Linearization of rational expressions

Intervalization: $\iota : (\texttt{exp}_\ell \times \mathcal{D}^\sharp) \to \texttt{exp}_\ell$

Intervalization flattens the expression into a single interval:

$$\iota(i_0 + \sum_k i_k \times \texttt{V}_k,\ \mathcal{X}^\sharp) \stackrel{\text{def}}{=} i_0 \oplus^\sharp \sum\nolimits_k^\sharp (i_k \otimes^\sharp \pi_k(\mathcal{X}^\sharp)).$$

Linearization without rounding errors: $\ell : (\texttt{exp} \times \mathcal{D}^\sharp) \to \texttt{exp}_\ell$

Defined by induction on the syntax of expressions:

- $\ell(\texttt{V}, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} [1,1] \times \texttt{V}$
- $\ell([a,b], \mathcal{X}^\sharp) \stackrel{\text{def}}{=} [a,b]$
- $\ell(e_1 + e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} \ell(e_1, \mathcal{X}^\sharp) \boxplus \ell(e_2, \mathcal{X}^\sharp)$
- $\ell(e_1 - e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} \ell(e_1, \mathcal{X}^\sharp) \boxminus \ell(e_2, \mathcal{X}^\sharp)$
- $\ell(e_1 / e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} \ell(e_1, \mathcal{X}^\sharp) \boxslash \iota(\ell(e_2, \mathcal{X}^\sharp), \mathcal{X}^\sharp)$
- $\ell(e_1 \times e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=}$ can be $\begin{cases} \text{either} & \iota(\ell(e_1, \mathcal{X}^\sharp), \mathcal{X}^\sharp) \boxtimes \ell(e_2, \mathcal{X}^\sharp) \\ \text{or} & \iota(\ell(e_2, \mathcal{X}^\sharp), \mathcal{X}^\sharp) \boxtimes \ell(e_1, \mathcal{X}^\sharp) \end{cases}$

Linearization of floating-point expressions

Rounding an affine interval form: (32-bit single precision)

- if the result is normalized: we have a relative error ε with magnitude 2^{-23}:

$$\varepsilon([a_0, b_0] + \textstyle\sum_k [a_k, b_k] \times \mathtt{V}_k) \stackrel{\text{def}}{=} \max(|a_0|, |b_0|) \times [-2^{-23}, 2^{-23}] + \textstyle\sum_k (\max(|a_k|, |b_k|) \times [-2^{-23}, 2^{-23}] \times \mathtt{V}_k)$$

- if the result is denormalized, we have an absolute error $\omega \stackrel{\text{def}}{=} [-2^{-159}, 2^{-159}]$.

$\Longrightarrow$ we sum these two sources of rounding errors

Linearization with rounding errors: $\ell : (\mathtt{exp}_f \times \mathcal{D}^\sharp) \to \mathtt{exp}_\ell$

$$\ell(e_1 \oplus_r e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} \ell(e_1, \mathcal{X}^\sharp) \boxplus \ell(e_2, \mathcal{X}^\sharp) \boxplus \varepsilon(\ell(e_1, \mathcal{X}^\sharp)) \boxplus \varepsilon(\ell(e_2, \mathcal{X}^\sharp)) \boxplus \omega$$

$$\ell(e_1 \otimes_r e_2, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} \iota(\ell(e_1, \mathcal{X}^\sharp), \mathcal{X}^\sharp) \boxtimes (\ell(e_2, \mathcal{X}^\sharp) \boxplus \varepsilon(\ell(e_2, \mathcal{X}^\sharp))) \boxplus \omega$$

...

Applications of the floating-point linearization

Soundness of the linearization

$\forall e,\ \forall \mathcal{X}^\sharp \in \mathcal{D}^\sharp,\ \forall \rho \in \gamma(\mathcal{X}^\sharp)$,
if $\mathcal{O} \notin \mathrm{E}[\![\, e \,]\!]\, \rho$, then $\mathrm{E}[\![\, e \,]\!]\, \rho \subseteq \mathrm{E}[\![\, \ell(e, \mathcal{X}^\sharp) \,]\!]\, \rho$

Application: $\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X}^\sharp$

- check that $\mathcal{O} \notin \mathrm{E}[\![\, e \,]\!]\, \rho$ for $\rho \in \gamma(\mathcal{X}^\sharp)$ with interval arithmetic
- compute $\mathrm{C}^\sharp[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X}^\sharp$ as $\mathrm{C}^\sharp[\![\, \mathtt{V} := \ell(e, \mathcal{X}^\sharp) \,]\!]\, \mathcal{X}^\sharp$
- (use $\mathrm{C}^\sharp[\![\, \mathtt{V} := [-Mf, Mf] \,]\!]\, \mathcal{X}^\sharp$ if $\mathcal{O} \in \mathrm{E}[\![\, e \,]\!]\, \rho$)

Sound floating-point polyhedra

Sound floating-point polyhedra

Algorithms to adapt: [Chen al. 08]

- linear programming:

$$\mathit{simplex}_f(\mathcal{X}^\sharp, \vec{\alpha}) \le \mathit{simplex}(\mathcal{X}^\sharp, \vec{\alpha})$$

$$\mathit{simplex}(\mathcal{X}^\sharp, \vec{\alpha}) \stackrel{\text{def}}{=} \min \{ \textstyle\sum_k \alpha_k \rho(\mathtt{V}_k) \mid \rho \in \gamma(\mathcal{X}^\sharp) \}$$

- Fourier-Motzkin elimination:

$$\mathit{Fourier}_f(\mathcal{X}^\sharp, \mathtt{V}_k) \Longleftarrow \mathit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_k)$$

$$\begin{aligned} \mathit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_k) &\stackrel{\text{def}}{=} \\ &\{ (\textstyle\sum_i \alpha_i \mathtt{V}_i \ge \beta) \in \mathcal{X}^\sharp \mid \alpha_k = 0 \} \cup \\ &\{ (-\alpha_k^-)c^+ + \alpha_k^+ c^- \mid \; c^+ = (\textstyle\sum_i \alpha_i^+ \mathtt{V}_i \ge \beta^+) \in \mathcal{X}^\sharp,\ \alpha_k^+ > 0, \\ &\qquad\qquad\qquad\qquad\quad\;\; c^- = (\textstyle\sum_i \alpha_i^- \mathtt{V}_i \ge \beta^-) \in \mathcal{X}^\sharp,\ \alpha_k^- < 0 \} \end{aligned}$$

Sound floating-point linear programming

Guaranteed linear programming: [Neumaier Shcherbina 04]

Goal: under-approximate $\mu = \min \{ \vec{c} \cdot \vec{x} \mid \mathbf{M} \times \vec{x} \leq \vec{b} \}$

knowing that $\vec{x} \in [\vec{x}_l, \vec{x}_h]$ (bounding-box for $\gamma(\mathcal{X}^\sharp)$).

- compute any approximation $\tilde{\mu}$ of the dual problem:

  $\tilde{\mu} \simeq \mu = \max \{ \vec{b} \cdot \vec{y} \mid {}^t\mathbf{M} \times \vec{y} = \vec{c},\ \vec{y} \leq \vec{0} \}$

  and the corresponding vector $\vec{y}$

  (e.g. using an off-the-shelf solver; $\tilde{\mu}$ may over-approximate or under-approximate μ)

- compute with intervals safe bounds $[\vec{r}_l, \vec{r}_h]$ for $\mathbf{A} \times \vec{y} - \vec{c}$:

  $[\vec{r}_l, \vec{r}_h] = ({}^t\mathbf{A} \otimes^\sharp \vec{y}) \ominus^\sharp \vec{c}$

  and then:

  $\nu = \inf((\vec{b} \otimes^\sharp \vec{y}) \ominus^\sharp ([\vec{r}_l, \vec{r}_h] \otimes^\sharp [\vec{x}_l, \vec{x}_h]))$

then: $\nu \leq \mu$.

Sound floating-point Fourier-Motzkin elimination

Given:

- $c^+ = (\sum_i \alpha_i^+ \mathtt{V}_i \geq \beta^+)$ with $\alpha_k^+ > 0$
- $c^- = (\sum_i \alpha_i^- \mathtt{V}_i \geq \beta^-)$ with $\alpha_k^- < 0$
- a bounding-box of $\gamma(\mathcal{X}^\sharp)$: $[\vec{x}_l, \vec{x}_h]$

We wish to compute $\sum_{i \neq k} \alpha_i \mathtt{V}_i \geq \beta$ in $\mathbb{F}$
implied by $(-\alpha_k^-)c^+ + \alpha_k^+ c^-$ in $\gamma(\mathcal{X}^\sharp)$.

- normalize c^+ and c^- using interval arithmetic:
  $$\begin{cases} \mathtt{V}_k + \sum_{i \neq k} (\alpha_i^+ \oslash^\sharp \alpha_k^+)\mathtt{V}_i \geq \beta^+ \oslash^\sharp \alpha_k^+ \\ -\mathtt{V}_k + \sum_{i \neq k} (\alpha_i^- \oslash^\sharp (-\alpha_k^-))\mathtt{V}_i \geq \beta^- \oslash^\sharp (-\alpha_k^-) \end{cases}$$
  (interval affine forms)
- add them using interval arithmetic:
  $$\sum_{i \neq k} [a_i, b_i]\mathtt{V}_i \geq [a_0, b_0]$$
  where $[a_i, b_i] = (\alpha_i^+ \oslash^\sharp \alpha_k^+) \ominus^\sharp (\alpha_i^- \oslash^\sharp \alpha_k^-)$,
  $[a_0, b_0] = (\beta^+ \oslash^\sharp \alpha_k^+) \ominus^\sharp (\beta^- \oslash^\sharp \alpha_k^-)$.

Sound floating-point Fourier-Motzkin elimination (cont.)

- linearize the interval affine form $\sum_{i \neq k} [a_i, b_i]\mathtt{V}_i \geq [a_0, b_0]$ into an affine form $\sum_{i \neq k} \alpha_i \mathtt{V}_i \geq \beta$

  we choose:

  - $\alpha_i \in [a_i, b_i]$
  - $\beta = \sup\left([a_0, b_0] \oplus^\sharp \bigoplus^\sharp_{i \neq k} (|\alpha_i \ominus^\sharp [a_i, b_i]|) \otimes^\sharp |[\vec{x}_l, \vec{x}_h]|\right)$

Soundness:

For all choices of $\alpha_i \in [a_i, b_i]$,
$\sum_{i \neq k} \alpha_i \mathtt{V}_k \geq \beta$ holds in $\mathit{Fourier}(\mathcal{X}^\sharp, \mathtt{V}_k)$.
(e.g. $\alpha_i = (a_i \oplus_n b_i) \oslash 2$)

Consequences of rounding

Precision loss:

- Projection:

$$\gamma(\mathit{Fourier}_f(\mathcal{X}^\sharp, \mathtt{V}_k)) \supsetneq \{\, \rho[\mathtt{V}_k \mapsto v] \mid v \in \mathbb{Q},\ \rho \in \gamma(\mathcal{X}^\sharp) \,\} = \mathrm{C}[\![\, \mathtt{V}_k := [-\infty, +\infty] \,]\!]\, \gamma(\mathcal{X}^\sharp)$$

- Order:

$$\mathcal{X}^\sharp \subseteq^\sharp \mathcal{Y}^\sharp \implies \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathcal{Y}^\sharp) \quad (\not\Leftarrow)$$

- Join:

$$\gamma(\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp) \supsetneq \mathit{ConvexHull}(\gamma(\mathcal{X}^\sharp) \cup \gamma(\mathcal{Y}^\sharp)) \quad (\neq)$$

Efficiency loss:

- cannot remove all redundant constraints

Abstraction summary

Floating-point polyhedra analyzer for floating-point programs

expression abstraction		**environment abstraction**
float expression e_f		
$\downarrow$ linearization		$\mathcal{P}(\mathcal{V} \to \mathbb{F})$
affine form e_ℓ in $\mathbb{Q}$		$\downarrow$ abstract domain
$\downarrow$ float implementation		polyhedra in $\mathbb{Q}$
affine form e_ℓ in $\mathbb{F}$	$\longrightarrow$	$\downarrow$ float implementation
		polyhedra in $\mathbb{F}$
		$\downarrow$ widening
		polyhedra in $\mathbb{F}$

Binary Representation Aware Domains

Integer Abstractions

Handling integer casts

Compute-through-overflow

```
signed char x, y; /* in [-1,1] */
(signed char) ( (unsigned char) x + (unsigned char) y )
```

Handling integer casts

Compute-through-overflow

```
signed char x, y; /* in [-1,1] */
(signed char) ( (unsigned char) x + (unsigned char) y )
```

Concrete semantics:

- conversion `signed char` $\rightarrow$ `unsigned char`
  $\Longrightarrow$ overflows, and maps $\{-1,0,1\}$ to $\{0,1,255\}$
- integer promotion: `unsigned char` $\rightarrow$ `int`
  $\Longrightarrow$ value preserving
- addition in `int`: $\Longrightarrow \{0,1,2,255,256,510\}$
- conversion `int` $\rightarrow$ `signed char`
  $\Longrightarrow$ overflows, and returns $\{-2,-1,0,1,2\}$

Handling integer casts

Compute-through-overflow

```
signed char x, y; /* in [-1,1] */
(signed char) ( (unsigned char) x + (unsigned char) y )
```

Interval semantics:

- conversion `signed char` → `unsigned char`
  $\Longrightarrow$ overflows, and maps $[-1,1]$ to $[0,255]$
  $\Longrightarrow$ all precision is lost
- the final result is $[-128,127]$

Issue:

the actual result $[-2,2]$ is representable in the interval domain
but the intermediate results are not! (not convex)

Handling integer casts

Compute-through-overflow

```
signed char x, y; /* in [-1,1] */
(signed char) ( (unsigned char) x + (unsigned char) y )
```

Modular interval domain:

invariants $[\ell, h] + k\mathbb{Z}$, $k \in \mathbb{N}$ (no hypothesis on bit-sizes of types)

- conversion `signed char` $\to$ `unsigned char`
  $\Longrightarrow$ overflows, and maps $[-1, 1]$ to $[-1, 1] + 256\mathbb{Z}$
- integer promotion: `unsigned char` $\to$ `int`
  $\Longrightarrow$ value preserving
- addition in `int`: $\Longrightarrow$ $[-2, 2] + 256\mathbb{Z}$
- conversion `int` $\to$ `signed char`
  $\Longrightarrow$ overflows, and returns $[-2, 2]$

Handling integer casts

Compute-through-overflow

```
signed char x, y; /* in [-1,1] */
(signed char) ( (unsigned char) x + (unsigned char) y )
```

Modular interval domain:

no Galois connection (no best abstraction)

- $[\ell, h] + 0\mathbb{Z}$ handed exactly as classic intervals
- $+^\sharp$, $-^\sharp$, $\times^\sharp$, $\cup^\sharp$ handed precisely
  e.g., $([\ell, h] + k\mathbb{Z}) +^\sharp ([\ell', h'] + k'\mathbb{Z}) = [\ell + \ell', h + h'] + \gcd(k, k')\mathbb{Z}$
- wrap-around: $\text{wrap}^\sharp([\ell, h] + k\mathbb{Z}, [a, b]) =$
  - $[\text{wrap}(\ell, [a, b]), \text{wrap}(h, [a, b])] + 0\mathbb{Z}$
    if $[\ell, h] + k\mathbb{Z}$ does not cross $a + (b - a)\mathbb{Z}$
  - $[\ell, h] + \gcd(k,\ b - a + 1)\mathbb{Z}$
    otherwise
- otherwise use interval information (reduced product)

Handling implicit integer casts

Code example

```
signed char x, y, z;
unsigned register r1, r2, r3;
r1 = x; r2 = y;
r3 = r1 + r2;
z = r3;
```

Handling implicit integer casts

Code example

```
signed char x, y, z;
unsigned register r1, r2, r3;
r1 = (unsigned) x; r2 = (unsigned) y;
r3 = r1 + r2;
z = (signed char) r3;
```

Use a pool of `register` variables to perform all computations
type mismatch $\Longrightarrow$ overflows and imprecision

- more difficult to detect by syntactic filters
  (implicit casts, computations spread on several instructions)
- can also be handled by modular integers
- also a common pattern in embedded software
  (manual register allocation, helps binary traceability)

Low-Level Memory Abstraction

Low-level memory access examples

Union

```
union {
  struct { uint8 al,ah,bl,bh } b;
  struct { uint16 ax,bx } w;
} r;
r.w.ax = 258;
if (r.b.al==2) r.b.al++;
```

Type-punning

```
uint8 buf[4] = { 1,2,3,4 };
uint32 i = *((uint32*)buf);
```

Fast copy

```
float a,b;
*((int*)&a) = *((int*)&b);
```

C standard: ill-typed programs, undefined behavior

In practice:

- there is no error
- the semantics is well-defined (ABI specification)

Low-level memory semantics

Concrete semantics: defined at the bit level

Abstract semantics:

decompose dynamically the memory into cells of scalar type:

- cell = variable, offset, and scalar type
- materialize new cells when needed by a dereference
  (possible reduction with existing cells)
- allow overlapping cells, with an intersection semantics

Orthogonality:

- memory domain: maps variables $\mathcal{V}$ to cells $\mathcal{C}$
- scalar domains: collections of independent cells $\mathcal{C} \to \texttt{Val}$

Pointers:

- **concrete:** semi-symbolic values: base $\in \mathcal{V}$ and offset $\in \mathbb{Z}$
- **abstraction:** Cartesian abstraction $\mathcal{P}(\mathcal{V}) \times \mathcal{P}(\mathbb{Z})$
  - keep $\mathcal{P}(\mathcal{V})$ in a pointer-specific domain
  - treat offsets as integer variables in numeric domains

Low-level memory example

0	1	2 ···

Union

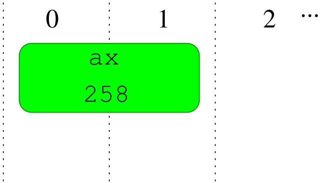
```
r.w.ax = 258;
if (r.b.al==2) r.b.al++;
```

initial state: no cell ($\top$)

Low-level memory example

Union

```
r.w.ax = 258;
if (r.b.al==2) r.b.al++;
```

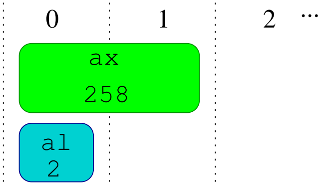


create `r.w.ax`, a `uint16` cell at offset 0

Low-level memory example

Union

```
r.w.ax = 258;
if (r.b.al==2) r.b.al++;
```

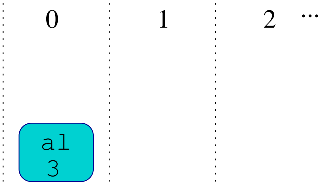


create `r.b.al`, a `uint8` cell at offset 0
initialized with: `r.w.ax` mod 256

Low-level memory example

Union

```
r.w.ax = 258;
if (r.b.al==2) r.b.al++;
```



modify cell `r.b.al`
destroy invalidated cell `r.w.ax`

Floating-Point Domains

Bit-level float manipulations

Cast

```
double cast(int i) {
  union { int i[2]; double d; } x, y;
  x.i[0] = 0x43300000; y.i[0] = x.i[0];
  x.i[1] = 0x80000000; y.i[1] = i ^ x.i[1];
  return y.d - x.d;
}
```

Bit-level float manipulations

Cast

```
double cast(int i) {
  union { int i[2]; double d; } x, y;
  x.i[0] = 0x43300000; y.i[0] = x.i[0];
  x.i[1] = 0x80000000; y.i[1] = i ^ x.i[1];
  return y.d - x.d;
}
```

- `0x43300000 0x80000000` represents $2^{52} + 2^{31}$
- `0x43300000 0x80000000 ^i` represents $2^{52} + 2^{31} + \texttt{i}$
- `y.d - x.d` equals `i`
  $\Longrightarrow$ cast from 32-bit signed int to 64-bit double

Justification:

- some CPUs miss the cast instruction (PowerPC)
- do not rely on the compiler to emulate it (code traceability)

Bit-level float manipulations

Cast

```
double cast(int i) {
  union { int i[2]; double d; } x, y;
  x.i[0] = 0x43300000; y.i[0] = x.i[0];
  x.i[1] = 0x80000000; y.i[1] = i ^ x.i[1];
  return y.d - x.d;
}
```

Analysis principle:

- memory domain: detects union usage
  smart initialization at materialization
  $\mathtt{y.d} = \mathit{dbl\_of\_word}(\mathtt{y.i}[0], \mathtt{y.i}[1])$
- new ad-hoc symbolic domain: maintains predicates
  - $\mathtt{V} = \mathtt{W\hat{}0x80000000}$ $\qquad (\mathtt{y.i}[1] = \mathtt{i\hat{}x.i}[1])$
  - $\mathtt{V} = \mathit{dbl\_of\_word}(\mathtt{0x43300000}, \mathtt{W})$ $\qquad (\mathtt{y.d})$

Bit-level float manipulations

Cast

```
double cast(int i) {
  union { int i[2]; double d; } x, y;
  x.i[0] = 0x43300000; y.i[0] = x.i[0];
  x.i[1] = 0x80000000; y.i[1] = i ^ x.i[1];
  return y.d - x.d;
}
```

reduction between intervals and predicates:

- predicates inferred by pattern-matching of expressions and values provided by intervals
  $(0x43300000, 0x80000000)$
- symbolic rewrite rules enrich intervals
  $(\mathtt{y.d} - \mathtt{x.d} \rightsquigarrow (\mathtt{double})\mathtt{i})$

easy to extent with new predicates and propagation rules!

More bit-level float manipulations

Extraction with a bit-mask

```
double d;
unsigned* p = (unsigned*) &d;
e = ((*p >> 20) & 0x7ff) - 1023;
```

Extraction with loop

```
double d, x = 1;
int e = 0;
if (d > 1)
  while (x < d) {
    e++; x *= 2;
  }
```

Both examples extract the exponent of a (normalized) 64-bit float.

Can be handled by:

- enriching the symbolic domain
  - $V = hi\_word(W)$
  - $V = 2^{W+i}, i \in \mathbb{Z}$
- adding new numeric domains
  - $V/W \in [\ell, h]$ (similar to difference-bound matrices)

Example application: Astrée

The Astrée static analyzer

Analyseur statique de programmes temps-réels embarqués

(static analyzer for real-time embedded software)

- developed at ENS (since 2001)

  B. Blanchet, P. Cousot, R. Cousot, J. Feret,
  L. Mauborgne, D. Monniaux, A. Miné, X. Rival

- industrialized and made commercially available by AbsInt (since 2009)

Astrée

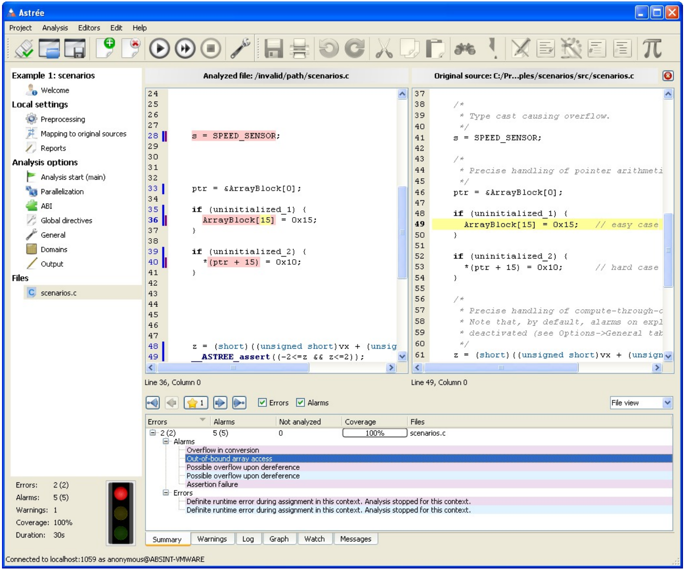
`www.astree.ens.fr`

AbsInt

`www.absint.com`

[Blanchet et al. 03]

The Astrée static analyzer



Specialized static analyzers

Design by refinement:

- focus on a specific family of programs and properties
- start with a fast and coarse analyzer (intervals)
- while the precision is insufficient (too many false alarms)
  - add new abstract domains (generic or application-specific)
  - refine existing domains (better transfer functions)
  - improve communication between domains (reductions)

$\Longrightarrow$ analyzer specialized for a (infinite) class of programs

- efficient and precise
- parametric (by end-users, to analyze new programs in the family)
- extensible (by developers, to analyze related families)

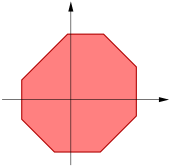
Astrée specialization

Specialized:

- for the analysis of run-time errors
  (arithmetic overflows, array overflows, divisions by 0, etc.)
- on embedded critical C software
  (no dynamic memory allocation, no recursivity)
- in particular on control / command software
  (reactive programs, intensive floating-point computations)
- intended for validation
  (analysis does not miss any error and tries to minimise false alarms)

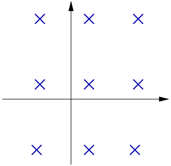
More Abstract Domain Examples

A few of the abstract domains used in Astrée.



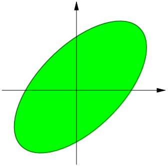
octagons

$\pm X \pm Y \leq c$



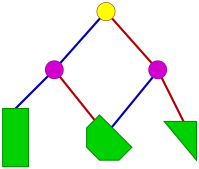
congruences

$X \equiv a\,[b]$

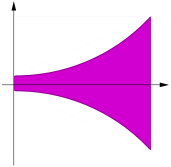


ellipsoids

digital filters

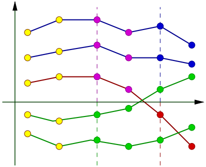


boolean decision trees



exponentials

$X \leq (1+\alpha)^{\beta t}$



trace partitions

Astrée applications (at ENS)

Airbus A340-300 (2003)

Airbus A380 (2004)

(model of) ESA ATV (2008)

- size: from 70 000 to 860 000 lines of C
- analysis time: from 45mn to $\simeq$40h
- alarm(s): 0 (proof of absence of run-time error)

The end