Non-linear and Floating-Point Abstractions

MPRI 2–6: Abstract Interpretation,
application to verification and static analysis

Antoine Miné

year 2014–2015

course 06
15 October 2014

Floating-point computations problematics

Two independent problems:

- **Analyze floating-point programs**

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

  Due to rounding, floating-point programs are highly non-linear
  $\Longrightarrow$ more general goal: **analyze non-linear expressions**

- **Implement an analyzer using floating-point numbers**

  goal: trade precision for efficiency

  exact rational arithmetics can be costly
  coefficients can grow large (polyhedra)
  $\Longrightarrow$ replace $\mathbb{Q}$ with $\mathbb{F}$

Combination: build a float analyzer for float programs.

Challenge: how to stay **sound**?

Outline

- **Floating-point numbers**
  - Concrete semantics
  - Floating-point intervals
    sound intervals for floats, implemented in floats
- **Linearization**
  - General framework for non-linear expressions
    more precise interval analyses
  - Application to floating-point expressions
    sound octagons for floats, implemented in floats
    sound polyhedra for floats, implemented in rationals
- **Floating-point polyhedra**
  - Constraint-only polyhedral algorithms
  - Sound floating-point approximate algorithms
    sound polyhedra for floats, implemented in floats
- Bibliography

Floating-point semantics

Floating-point numbers

Real computers do not know about $\mathbb{Q}$ and $\mathbb{R}$.
They use limited-precision floating-point numbers $\mathbb{F}$.

IEEE 754-1985 standard is the most widespread format.

(supported by most processors and programming languages)

IEEE Binary representation: a number is a triple $\langle s, e, f \rangle$

- a 1-bit sign s,
- a e-bit exponent e, with a bias (e represents $e - \text{bias}$),
- a p-bit fraction $f = .b_1 \ldots b_p$, (f represents $\sum_i 2^{-i} b_i$).

IEEE format examples given by the choice of e, bias, p:

32-bit single precision *float*: $\begin{cases} \text{e} = 8, \\ \text{bias} = 127, \\ \text{p} = 23. \end{cases}$

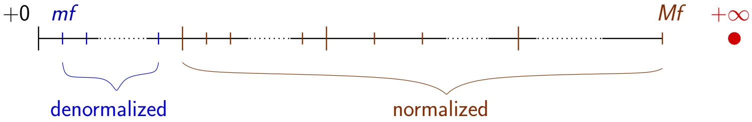
Other widespread formats: 64-bit *double*, 80-bit *double extended*, 128-bit *quad*.

Floating-point representation

Semantics $\langle s, e, f \rangle$ represents either:

- a normalized number: $(-1)^s \times 2^{e-\text{bias}} \times 1.f$ (if $1 \leq e \leq 2^{\text{e}} - 2$);
- a denormalized number: $(-1)^s \times 2^{1-\text{bias}} \times 0.f$ (if $e = 0,\ f \neq 0$);
- $+0$ or -0 (if $e = 0,\ f = 0$);
- $+\infty$ or $-\infty$ (if $e = 2^{\text{e}} - 1,\ f = 0$);
- an error code *NaN* (if $e = 2^{\text{e}} - 1,\ f \neq 0$).

Visual representation (positive part)



$mf \stackrel{\text{def}}{=} 2^{1-\text{bias}-\text{p}}$ smallest positive

$Mf \stackrel{\text{def}}{=} (2 - 2^{-\text{p}}) \times 2^{2^{\text{e}}-\text{bias}-2}$ largest non-∞

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*.

Simplified semantics:

- overflows and NaNs halt the program with an error Ω,
- rounding and underflow are not errors,
- we do not distinguish between $+0$ and -0.
  (in C, $+0$ == -0; however, $1/+0 = +\infty$ while $1/-0 = -\infty$)

$\Longrightarrow$ variable values live in a finite subset $\mathbb{F}$ of $\mathbb{R}$,
expression values live in $\mathbb{F} \cup \{\Omega\}$.

Floating-point computations (cont.)

Floating-point expressions $\texttt{exp}^{\mathbb{F}}$

The syntax of expression is now:

$$
\begin{array}{rclll}
\texttt{exp}^{\mathbb{F}} & ::= & [c, c'] & \text{constant interval } c, c' \in \mathbb{F} \\
& | & \texttt{V} & \text{variable } \texttt{V} \in \mathbb{V} \\
& | & \ominus\ \texttt{exp}^{\mathbb{F}} & \text{negation} \\
& | & \texttt{exp}^{\mathbb{F}} \odot \texttt{exp}^{\mathbb{F}} & \text{operator } \odot \in \{ \oplus, \ominus, \otimes, \oslash \}
\end{array}
$$

(we use circled operators: $\oplus, \ldots$ to distinguish them from operators in $\mathbb{R}$: $+, \ldots$)

Concrete semantics of expressions

Semantics of rounding: $R_r \colon \mathbb{R} \to \mathbb{F} \cup \{\Omega\}$.

rounding modes r: towards $+\infty$, $-\infty$, 0, or to-nearest n.

Example definitions:

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

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

Notes:

- $\forall x, r \colon R_r(x) \in [R_{-\infty}(x), R_{+\infty}(x)]$
  (actually: $\forall x, r \colon R_r(x) \in \{\, R_{-\infty}(x), R_{+\infty}(x) \,\}$)
- $\forall r \colon R_r$ is monotonic

Concrete semantics of expressions (cont.)

$\mathsf{E}[\![\, e \,]\!] : (\mathbb{V} \to \mathbb{F}) \to \mathcal{P}(\mathbb{F} \cup \{\Omega\})$ (expression semantics)

Each operator is evaluated in $\mathbb{R}$ and then rounded using R_r.

$$
\begin{array}{ll}
\mathsf{E}[\![\, \mathtt{V} \,]\!]\, \rho & \stackrel{\text{def}}{=} \{\, \rho(\mathtt{V}) \,\} \\
\mathsf{E}[\![\, [c, c'] \,]\!]\, \rho & \stackrel{\text{def}}{=} \{\, x \in \mathbb{F} \mid c \leq x \leq c' \,\} \\
\mathsf{E}[\![\, \ominus\, e \,]\!]\, \rho & \stackrel{\text{def}}{=} \{\, -x \mid x \in \mathsf{E}[\![\, e \,]\!]\, \rho \cap \mathbb{F} \,\} \cup \{\, \Omega \mid \text{if } \Omega \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\} \\
\mathsf{E}[\![\, e_1 \odot_r e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=}
\end{array}
$$

$$
\begin{array}{l}
\{\, R_r(x_1 \cdot x_2) \mid x_1 \in \mathsf{E}[\![\, e_1 \,]\!]\, \rho \cap \mathbb{F},\ x_2 \in \mathsf{E}[\![\, e_2 \,]\!]\, \rho \cap \mathbb{F} \,\} \cup \\
\{\, \Omega \mid \ \text{if } \Omega \in \mathsf{E}[\![\, e_1 \,]\!]\, \rho \cup \mathsf{E}[\![\, e_2 \,]\!]\, \rho \,\} \\
\{\, \Omega \mid \ \text{if } 0 \in \mathsf{E}[\![\, e_2 \,]\!]\, \rho \text{ and } \odot = \oslash \,\}
\end{array}
$$

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

$$
\begin{array}{ll}
\mathsf{C}[\![\, \mathtt{V} := e \,]\!]\, \mathbb{V} & \stackrel{\text{def}}{=} \{\, \rho[\, \mathtt{V} \mapsto v \,] \mid \rho \in \mathcal{X},\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho \cap \mathbb{F} \,\} \\
 & \quad \cup \{\, \Omega \mid \text{if } \Omega \in \mathsf{E}[\![\, e \,]\!]\, \mathcal{X} \,\} \\
\mathsf{C}[\![\, e \bowtie 0 \,]\!]\, \mathcal{X} & \stackrel{\text{def}}{=} \{\, \rho \mid \rho \in \mathcal{X},\ \exists v \in \mathsf{E}[\![\, e \,]\!]\, \rho \cap \mathbb{F} \colon v \bowtie 0 \,\} \\
 & \quad \cup \{\, \Omega \mid \text{if } \Omega \in \mathsf{E}[\![\, e \,]\!]\, \mathcal{X} \,\}
\end{array}
$$

Floating-point interval domain

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

Expression semantics: $\mathsf{E}^\sharp[\![\, \exp^{\mathbb{F}} \,]\!] : (\mathbb{V} \to \mathcal{B}^\sharp) \to \mathcal{B}^\sharp$

Computed by induction using:

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

- 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 arithmetics (efficiency).

Error management

If some bound in $\mathsf{E}^\sharp[\![\, \exp^{\mathbb{F}} \,]\!]$ evaluates to Ω, we

- report the error to the user, and
- continue the evaluation with $[-Mf, Mf]$ (errors are not propagated).

Floating-point analysis example

filter with reinitialisation

```
Z:=0;
while 1=1 do
  if [0,1]=1 then Z:=[-10,10] fi;
  Z:=(0.3 ⊗ Z) ⊕ [-10,10]
done
```

In $\mathbb{R}$, we would have $|\mathtt{Z}| < 10/0.7$.

Using floats, $|\mathtt{Z}|$ is bounded by $B = R_{+\infty}(10/0.7)$.

Interval analysis:

A widening with thresholds finds that $|\mathtt{Z}| \leq \min\,\{\, x \in T \mid x \geq B \,\}$.

The absence of overflow is proved if T has a value larger than B.

Issues with relational domains

Relational domains exploit many properties: associativity, distributivity, . . . ; they are true in $\mathbb{Q}$ and $\mathbb{R}$, but not true in $\mathbb{F}$!

Replacing $(\mathbb{Q}, +, -, \times, /)$ with $(\mathbb{F}, \oplus, \ldots, \otimes, \oslash)$ in the algorithms is **not sound**.

Example: (DBM closure)

$$(X - Y \leq c) \wedge (Y - Z \leq d) \Longrightarrow (X - Z \leq c + d)$$

$$(X \ominus Y \leq c) \wedge (Y \ominus Z \leq d) \not\Longrightarrow (X \ominus Z \leq c \oplus d)$$

$$(10^{22} \ominus 1.000000019 \cdot 10^{38}) \oplus (1.000000019 \cdot 10^{38} \ominus -10^{22}) = 0 \neq 10^{23}$$

Solution: [Mine04]

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

Linearization

Abstraction framework

Most relational domains can only deal with linear expressions.
How can we abstract non-linear assignments such as $\mathtt{X} := \mathtt{Y} \times \mathtt{Z}$?

Idea: replace $\mathtt{Y} \times \mathtt{Z}$ with a sound linear approximation.

(float expressions are also highly non-linear, when expressed in $\mathbb{Q}$)

Framework:

We define an approximation preorder $\preceq$ on expressions:

$$R \models e_1 \preceq e_2 \stackrel{\text{def}}{\iff} \forall \rho \in R\colon \mathsf{E}[\![\, e_1 \,]\!]\, \rho \subseteq \mathsf{E}[\![\, e_2 \,]\!]\, \rho.$$

Soundness properties if $\gamma(\mathcal{X}^\sharp) \models e \preceq e'$ then:

- $\mathsf{C}[\![\, \mathtt{V} := e \,]\!]\, \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathsf{C}^\sharp[\![\, \mathtt{V} := e' \,]\!]\, \mathcal{X}^\sharp)$
- $\mathsf{C}[\![\, e \bowtie 0 \,]\!]\, \gamma(\mathcal{X}^\sharp) \subseteq \gamma(\mathsf{C}^\sharp[\![\, e' \bowtie 0 \,]\!]\, \mathcal{X}^\sharp)$
- $\gamma(\mathcal{X}^\sharp) \cap (\overleftarrow{\mathsf{C}}[\![\, \mathtt{V} := e \,]\!]\, \gamma(\mathcal{R}^\sharp)) \subseteq \gamma(\overleftarrow{\mathsf{C}}^\sharp[\![\, \mathtt{V} := e' \,]\!]^\sharp(\mathcal{X}^\sharp, \mathcal{R}^\sharp))$

$\Longrightarrow$ we can now use e' in the abstract instead of e.

Linearization

In practice, we put expressions into affine interval form:

$$\texttt{exp}_\ell : [a_0, b_0] + \sum_k [a_k, b_k] \times \texttt{V}_\texttt{k}$$

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 $C^\sharp[\![\, \texttt{V} := e_\ell \,]\!]$ and $C^\sharp[\![\, e_\ell \bowtie 0 \,]\!]$ in most domains

Linearization (cont.)

Operations on affine interval forms

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

Noting i_k the interval $[a_k, b_k]$ and using interval operations $+^\sharp_b$, $-^\sharp_b$, $\times^\sharp_b$, $/^\sharp_b$ (e.g., $[a,b] +^\sharp_b [c,d] = [a+c, b+d]$):

- $(i_0 + \sum_k i_k \times \mathtt{V_k}) \boxplus (i'_0 + \sum_k i'_k \times \mathtt{V_k}) \stackrel{\text{def}}{=} (i_0 +^\sharp_b i'_0) + \sum_k (i_k +^\sharp_b i'_k) \times \mathtt{V_k}$
- $i \boxtimes (i_0 + \sum_k i_k \times \mathtt{V_k}) \stackrel{\text{def}}{=} (i \times^\sharp_b i_0) + \sum_k (i \times^\sharp_b i_k) \times \mathtt{V_k}$
- ...

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

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 (cont.)

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

Flattens the expression into a single interval:

$$\iota(i_0 + \sum_k (i_k \times \mathtt{V_k}),\ \mathcal{X}^\sharp) \stackrel{\text{def}}{=} i_0 \ +_b^\sharp\ \sum_{b,\,k}^\sharp (i_k \times_b^\sharp \pi_k(\mathcal{X}^\sharp)).$$

Linearization $\ell : (\text{exp} \times \mathcal{D}^\sharp) \to \text{exp}_\ell$

Defined by induction on the syntax of expressions:

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

Linearization application

Property soundness of the linearization:

For any abstract domain $\mathcal{D}^\sharp$, any $\mathcal{X}^\sharp \in \mathcal{D}^\sharp$ and $e \in \text{exp}$, we have:

$$\gamma(\mathcal{X}^\sharp) \models e \preceq \ell(e, \mathcal{X}^\sharp)$$

Remarks:

- $\mathcal{X}^\sharp$ is used in π_k by ι; hence $\preceq$ holds only wrt. $\gamma(\mathcal{X}^\sharp)$,
- ℓ results in a loss of precision,
- ℓ is not monotonic for $\preceq$.
  (e.g., $\ell(\texttt{V}/\texttt{V}, \texttt{V} \mapsto [1, +\infty]) = [0, 1] \times \texttt{V} \not\preceq 1$)

Application to the octagon domain

```
Y:=[0,+∞];
T:=[-1,1];
X:=T×Y
```

- $\texttt{T} \times \texttt{Y}$ is linearized as $[-1, 1] \times \texttt{Y}$,
- we can prove that $|\texttt{X}| \leq \texttt{Y}$.

Linearization application (cont.)

Application to the interval domain

$C^\sharp[\![\, \mathtt{V} := \ell(e, \mathcal{X}^\sharp) \,]\!]\, \mathcal{X}^\sharp$ is always more precise than $C^\sharp[\![\, \mathtt{V} := e \,]\!]\, \mathcal{X}^\sharp$

ℓ simplifies symbolically variables occurring several times.

Example: $\mathtt{X} := 2 \times \mathtt{V} - \mathtt{V}$, where $\mathtt{V} \in [a, b]$:

- using vanilla intervals:

$$E^\sharp[\![\, 2 \times \mathtt{V} - \mathtt{V} \,]\!]\, (\mathcal{X}^\sharp) = 2 \times^\sharp_b [a, b] -^\sharp_b [a, b] = [2a - b, 2b - a],$$

- after linearization $\ell(2 \times \mathtt{V} - \mathtt{V}, \mathcal{X}^\sharp) = \mathtt{V}$, so
  $E^\sharp[\![\, \ell(2 \times \mathtt{V} - \mathtt{V}, \mathcal{X}^\sharp) \,]\!]\, \mathcal{X}^\sharp = [a, b]$

  strictly more precise than $[2a - b, 2b - a]$ when $a \neq b$.

Floating-point linearization

Floating-point linearization

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

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

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

- if the result is denormalized, we have an absolute error $\omega \stackrel{\text{def}}{=} [-2^{-149}, 2^{-149}]$.

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

Linearization: $\ell^{\mathbb{F}} : (\exp^{\mathbb{F}} \times \mathcal{D}^\sharp) \to \exp_\ell$

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

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

etc.

Soundness 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 $\Omega \notin \mathsf{E}[\![\, e \,]\!]\, \rho$, then $\mathsf{E}[\![\, e \,]\!]\, \rho \subseteq \mathsf{E}[\![\, \ell^{\mathbb{F}}(e, \mathcal{X}^\sharp) \,]\!]\, \rho$

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

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

Example applications

- Improving the interval domain using symbolic simplification.

  Example:

  $Z := X \ominus (0.25 \otimes X)$ is linearized into
  $Z := ([0.749\cdots, 0.750\cdots] \times X) + 2.35\cdots 10^{-38} \times [-1, 1]$.

  If $X \in [-1, 1]$, we find $|Z| \leq 0.750\cdots$
  (instead of $|Z| \leq 1.25\cdots$).

- Allows using relational domains (octagons, etc.)

  Example: floating-point version of the rate limiter

  (single precision)

  The bound of the output $|Y|$ is the smallest threshold larger than 144.00005 (instead of 144).

Floating-point implementation

Goal: implement abstract domains using floating-point numbers

- more efficient (especially to analyse floating-point programs),
- rounding errors in the algorithms may cause unsoundness!

Simple solution:

round upper-bounds toward $+\infty$, lower bounds toward $-\infty$

Works for:

- intervals ($\oplus_b^\sharp, \ominus_b^\sharp, \otimes_b^\sharp, \ldots$)
- linearization into $\texttt{exp}_\ell$ (based on interval computations)
- octagons (replace $a + b$ with $R_{+\infty}(a + b)$)
- not polyhedra

Constraint-only polyhedra

Reminders on the double description method

Two representations for polyhedra:

- Constraint representation $\langle \mathbf{M}, \vec{C} \rangle$

  $\gamma(\langle \mathbf{M}, \vec{C} \rangle) \stackrel{\text{def}}{=} \{ \vec{V} \mid \mathbf{M} \times \vec{V} \geq \vec{C} \}$

  where $\mathbf{M} \in \mathbb{I}^{m \times n}$ and $\vec{C} \in \mathbb{I}^m$

- Generator representation $[\mathbf{P}, \mathbf{R}]$

  $\gamma([\mathbf{P}, \mathbf{R}]) \stackrel{\text{def}}{=} \left\{ \left(\sum_{j=1}^{p} \alpha_j \vec{P}_j \right) + \left(\sum_{j=1}^{r} \beta_j \vec{R}_j \right) \mid \forall j, \alpha_j, \beta_j \geq 0 \text{: } \sum_{j=1}^{p} \alpha_j = 1 \right\}$

  where $\vec{P}_1, \ldots, \vec{P}_p \in \mathbb{I}^{n \times p}$ are points and $\vec{R}_1, \ldots, \vec{R}_r \in \mathbb{I}^{n \times r}$ are rays.

Benefits:

- operators are easy given the right representation
- only one complex algorithm:
  Chernikova's conversion algorithm

Constraint-only polyhedron domain

It is possible to use only the constraint representation:

- avoids the cost of Chernikova's algorithm,
- avoids exponential generator systems (hypercubes).

The core operations are: projection and redundancy removal.

Projection: using Fourier-Motzkin elimination

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

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

we then have:

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

Constraint-only polyhedron domain (cont.)

Fourier causes a quadratic growth in constraint number.
Most such constraints are redundant.

Redundancy removal: using linear programming [Schr86]

Let *simplex*$(\mathcal{Y}^\sharp, \vec{v}) \stackrel{\text{def}}{=} \min \{ \vec{v} \cdot \vec{y} \mid \vec{y} \in \gamma(\mathcal{Y}^\sharp) \}$

If $c = (\vec{\alpha} \cdot \vec{\mathrm{V}} \geq \beta) \in \mathcal{X}^\sharp$ and $\beta \leq \textit{simplex}(\mathcal{X}^\sharp \setminus \{c\}, \vec{\alpha})$,
then c can be safely removed from $\mathcal{X}^\sharp$.
(iterate over all constraints)

Note: running *simplex* many times can be become costly

- use fast syntactic checks first,
- check against the bounding-box first.

Constraint-only polyhedron domain (cont.)

Constraint-only abstract operators:

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

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

$\mathcal{X}^\sharp \cap^\sharp \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \mathcal{X}^\sharp \cup \mathcal{Y}^\sharp$ (join constraint sets)

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

$C^\sharp[\![\sum_i \alpha_i \mathtt{V}_i + \beta \geq 0]\!] \mathcal{X}^\sharp$ as before

$C^\sharp[\![\mathtt{V}_j := \sum_i \alpha_i \mathtt{V}_i + \beta]\!] \mathcal{X}^\sharp$ as before

Constraint-only polyhedron domain (cont.)

Constraint-only convex hull:

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

Formally:

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

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

[Beno96]

Floating-point polyhedra

Sound floating-point polyhedra

Algorithms to adapt: [Chen08]

Design sound approximate floating-point algorithms *simplex*$_f$ and *Fourier*$_f$.

- linear programming:

$$simplex_f(\mathcal{X}^\sharp, \vec{\alpha}) \leq simplex(\mathcal{X}^\sharp, \vec{\alpha})$$

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

- Fourier-Motzkin elimination:

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

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

Sound floating-point linear programming

Guaranteed linear programming: [Neum04]

Goal: under-approximate $\mu = \min \{ \vec{c} \cdot \vec{x} \,|\, \mathbf{M} \times \vec{x} \leq \vec{b} \}$
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:
  $\tilde{\mu} \simeq \mu = \max \{ \vec{b} \cdot \vec{y} \,|\, {}^t\mathbf{M} \times \vec{y} = \vec{c},\ \vec{y} \leq \vec{0} \}$
  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 $\mathbf{M} \times \vec{y} - \vec{c}$:
  $[\vec{r}_l, \vec{r}_h] = ({}^t\mathbf{M} \otimes_b^\sharp \vec{y}) \ominus_b^\sharp \vec{c}$

  and then:
  $\nu = \inf((\vec{b} \otimes_b^\sharp \vec{y}) \ominus_b^\sharp ([\vec{r}_l, \vec{r}_h] \otimes_b^\sharp [\vec{x}_l, \vec{x}_h]))$

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 arithmetics:

$$\begin{cases} \mathtt{V}_k + \sum_{i \neq k} (\alpha_i^+ \oslash_b^\sharp \alpha_k^+) \mathtt{V}_i \geq \beta^+ \oslash_b^\sharp \alpha_k^+ \\ -\mathtt{V}_k + \sum_{i \neq k} (\alpha_i^- \oslash_b^\sharp (-\alpha_k^-)) \mathtt{V}_i \geq \beta^- \oslash_b^\sharp (-\alpha_k^-) \end{cases}$$

  (interval affine forms)

- add them using interval arithmetics:

$$\sum_{i \neq k} [a_i, b_i] \mathtt{V}_i \geq [a_0, b_0]$$

  where $[a_i, b_i] = (\alpha_i^+ \oslash_b^\sharp \alpha_k^+) \ominus_b^\sharp (\alpha_i^- \oslash_b^\sharp \alpha_k^-)$,
  $[a_0, b_0] = (\beta^+ \oslash_b^\sharp \alpha_k^+) \ominus_b^\sharp (\beta^- \oslash_b^\sharp \alpha_k^-)$.

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

- linearize the interval linear form into $\sum_{i \neq k} \alpha_i \mathtt{V}_i \geq \beta$ where

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

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. $\alpha_i = (a_i \oplus b_i) \oslash 2$)

Consequences of rounding

Precision loss:

- Projection:

$$\begin{array}{rcl} \gamma(\mathit{Fourier}_f(\mathcal{X}^\sharp, \mathtt{V}_k)) & \supseteq & \{\, \rho[\mathtt{V}_k \mapsto v] \mid v \in \mathbb{Q},\ \rho \in \gamma(\mathcal{X}^\sharp) \,\} \\ & = & \mathsf{C}[\![\, \mathtt{V}_k := [-\infty, +\infty] \,]\!]\, \gamma(\mathcal{X}^\sharp) \end{array}$$

- Order:

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

- Join:

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

Efficiency loss:

- cannot remove all redundant constraints

Floating-point polyhedra widening

Widening $\triangledown$:

$$\mathcal{X}^\sharp \triangledown \mathcal{Y}^\sharp \stackrel{\text{def}}{=} \{ c \in \mathcal{X}^\sharp \,|\, \mathcal{Y}^\sharp \subseteq^\sharp \{ c \} \}$$

(drop $\{ c \in \mathcal{Y}^\sharp \,|\, \exists c' \in \mathcal{X}^\sharp \colon \mathcal{X}^\sharp =^\sharp (\mathcal{X}^\sharp \setminus c') \cup \{ c \} \}$
as $\mathcal{X}^\sharp$ and $\mathcal{Y}^\sharp$ may have redundant constraints)

Stability improvement:

robust strategies to choose $\alpha_i \in [a_i, b_i]$ during Fourier-Motzkin:

- choose simple α_i (e.g., integer nearest $(a_i \oplus b_i)/2$)
- reuse the same (or a multiple of) α_i used for other variables

Abstraction summary

Floating-point polyhedra analyzer for floating-point programs

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

Bibliography

Bibliography

[Beno96] **F. Benoy & A. King**. *Inferring argument size relationships with CLP(R).* In In Proc. of LOPSTR'96, LNCS 1207, 204–223. Springer, 1996.

[Chen08] **L. Chen, A. Miné & P. Cousot**. *A sound floating-point polyhedra abstract domain.* In Proc. APLAS'08, LNCS 5356, 3–18, Springer, 2008.

[Cous78] **P. Cousot & N. Halbwachs**. *Automatic discovery of linear restraints among variables of a program.* In Proc. POPL'78, 84–96, ACM, 1978.

[Jean09] **B. Jeannet & A. Miné**. *Apron: A library of numerical abstract domains for static analysis.* In Proc. CAV'09, LNCS 5643, 661–667, Springer, 2009, `http://apron.cri.ensmp.fr/library`.

Bibliography (cont.)

[Mine01b] **A. Miné**. *The octagon abstract domain.* In Proc. AST'01, 310–319, IEEE, 2001.

[Mine04] **A. Miné**. *Relational abstract domains for the detection of floating-point run-time errors.* In Proc. ESOP'04, LNCS 2986, 3–17, Springer, 2004.

[Neum04] **A. Neumaier & O. Shcherbina**. *Safe bounds in linear and mixed-integer linear programming.* In Math. Program., 99(2):283–296, 2004.

[Schr86] **A. Schrijver**. *Theory of linear and integer programming.* In John Wiley & Sons, Inc., 1986.