PhD. Defense

Weakly Relational Numerical Abstract Domains

Domaines numériques abstraits faiblement relationnels

Antoine Miné

École Normale Supérieure, Paris

December 6-th, 2004
Introduction

Main Goal:

• discover properties on the **numerical** variables of a program,

• **statically**, at compile-time,

• **automatically**, without human interaction.

Applications of Numerical Properties:

• Check for illegal arithmetic operations: overflow, division by zero.  
  (Ariane 5 explosion on June the 4-th 1996 $\Rightarrow$ $500$ M loss)

• Check for out-of-bound array or pointer accesses.

• Optimisation, debugging information inference.

• Parameters to non-numerical analyses.  
  (pointer analyses, parametric predicate abstractions, etc.)
Overview

♦ Formal framework, previous work, motivation for our work.


♦ Improving the precision using generic symbolic manipulation techniques.

♦ Dealing with floating-point semantics.

♦ Application within the Astrée analyser and experimental results.
(Simplified) Formal Framework
Language Syntax

For the sake of presentation:

- one data-type: **scalars** in $\mathbb{I}$, where $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$,
- no procedure,
- a **finite, fixed** set of variables: $\mathcal{V}$.

♦ **instructions** $\mathcal{I} ::= X \leftarrow E$ assignment to $X \in \mathcal{V}$
  | $E \bowtie 0$? test $\bowtie \in \{=, \leq, \ldots\}$

♦ **expressions** $E ::= [a, b]$ interval $a \in \mathbb{I} \cup \{-\infty\}$, $b \in \mathbb{I} \cup \{+\infty\}$,
  | $X$ variable $X \in \mathcal{V}$
  | $-E$ unary operator
  | $E \diamond E$ binary operators $\diamond \in \{+, \times, \ldots\}$

**Note:** $[a, b]$ models a **non-deterministic** choice within an interval.
**Semantics**

**Environments:** maps $\rho \in (\mathcal{V} \to \mathbb{I})$.

**Expression Semantics:** $\llbracket E \rrbracket: (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$

$E$ maps environments to sets of numerical values:

- $\llbracket [a, b] \rrbracket(\rho) \overset{\text{def}}{=} \{ c \in \mathbb{I} \mid a \leq c \leq b \}$,
- $\llbracket X \rrbracket(\rho) \overset{\text{def}}{=} \{ \rho(X) \}$,
- $\llbracket e_1 + e_2 \rrbracket(\rho) \overset{\text{def}}{=} \{ v_1 + v_2 \mid v_1 \in \llbracket e_1 \rrbracket(\rho), \ v_2 \in \llbracket e_2 \rrbracket(\rho) \}$,
  etc.

**Instruction Semantics:** $\{ I \}: \mathcal{P}(\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$

A transfer function defines a relation between environments:

- $\{ X \leftarrow e \}(R) \overset{\text{def}}{=} \{ \rho[ X \leftarrow v ] \mid \rho \in R, \ v \in \llbracket e \rrbracket(\rho) \}$,
- $\{ e \triangleright 0 \ ? \}(R) \overset{\text{def}}{=} \{ \rho \in R \mid \exists v \in \llbracket e \rrbracket(\rho) \text{ such that } v \triangleright 0 \}$. 

PhD. Defense - Weakly Relational Numerical Abstract Domains - Antoine Miné
Reachability Semantics

Given a control-flow graph \((L, e, I)\):

- \(L\): program points
- \(e \in L\): entry point
- \(I \subseteq L \times T \times L\): arcs

we seek to compute the reachability semantics, the smallest solution of:

\[
\mathcal{X}_l = \begin{cases} 
(V \rightarrow \mathbb{I}) & \text{if } l = e \quad \text{(initial state)} \\
\bigcup_{(l',i,l) \in I} \{i\}(\mathcal{X}_{l'}) & \text{if } l \neq e \quad \text{(transfer function)}
\end{cases}
\]

that gathers all possible environments at each program points.

**Problem:** This is **not computable** in general.

\(\implies\) we will compute sound over-approximations of the \(\mathcal{X}_l\)…
Abstract Interpretation

Abstract Interpretation:
General theory of sound approximations of semantics [Cousot78].

Numerical Abstract Domain:

- **computer-representable** set $\mathcal{D}^\#$ of abstract values, together with:
  - a concretisation: $\gamma: \mathcal{D}^\# \rightarrow \mathcal{P}(\mathcal{V} \rightarrow \mathcal{I})$,
  - a partial order: $\sqsubseteq^\#, \perp^\#, \top^\#$,
  - sound, effective abstract transfer functions $\{\mathcal{I}\}^\#$: $(\{\mathcal{I}\} \circ \gamma)(\mathcal{X}^\#) \subseteq (\gamma \circ \{\mathcal{I}\}^\#)(\mathcal{X}^\#)$,
  - a sound, effective abstract union $\cup^\#$: $\gamma(\mathcal{X}^\#) \cup \gamma(\mathcal{Y}^\#) \subseteq \gamma(\mathcal{X}^\# \cup^\# \mathcal{Y}^\#)$,
  - effective extrapolation operators $\nabla, \triangle$ if $\mathcal{D}^\#$ has infinite chains.

$\implies$ we can perform a reachability analysis in $L \rightarrow \mathcal{D}^\#$ soundly.

There does not exist an all-purpose abstract domain.
We need a fine control on both the **semantic** and **algorithmic** aspects!
Existing Numerical Abstract Domains

Before this work, the two most used numerical abstract domains were:

Intervals (1976)
\[ \bigwedge_i (X_i \in [a_i, b_i]) \]

Polyhedra (1978)
\[ \bigwedge_j (\sum_i \alpha_{ij} X_i \leq \beta_j) \]

- non relational
- linear cost

- relational
- unbounded cost

There were other domains, but no domain “in-between” these two.
The Need for Relational Domains

Example:

I := 10
V := 0
while • (I ≥ 0) {
    I := I − 1
    if (random()) { V := V + 1 }
} • // here I = −1 and 0 ≤ V ≤ 11

To prove that V ≤ 11 at •, we need to prove the relational loop invariant V + I ≤ 10 at •.

Other applications:
- analysis of programs with symbolic parameters,
- modular analysis of procedures, (out of context)
- inference of non-uniform non-numerical invariants.  (e.g., pointer analysis)
Weakly Relational Abstract Domains

New abstract domains introduced in this PhD:

• zone abstract domain,
• octagon abstract domain,
• zone congruence abstract domain.
The Zone Abstract Domain

**Simplest** of our three domains, but **characteristic** in its construction.

**Zones** enrich intervals with invariants of the form:

\[ \bigwedge_{i,j} (V_i - V_j \leq c_{ij}) \quad c_{ij} \in I \]

The zone abstract domain features:

- a precision between the interval and polyhedron domains; **relational** invariants,
- a **quadratic** memory cost and **cubic** worst-case time cost.

Zones are used in the model-checking of timed automata and Petri nets but they need many new abstract operators to suit Abstract Interpretation needs.
**Zone Representation**

**Difference Bound Matrices:** (DBMs)

- matrix of size \((n + 1) \times (n + 1)\) with elements in \(\bar{\mathbb{I}} \, \text{def} \, \mathbb{I} \cup \{+\infty\}\):
  - \(m_{ij} \neq +\infty\) is an upper bound for \(V_j - V_i\),
  - \(m_{ij} = +\infty\) means that \(V_j - V_i\) is unbounded,
  - \(m_{i0}, m_{0j}\) encode unary constraints: \(-V_i \leq m_{i0}, V_j \leq m_{0j}\),

- \(\gamma(m) \, \text{def} \, \{ (v_1, \ldots, v_n) \in \mathbb{I} \mid \forall i, j, v_j - v_i \leq m_{ij}, v_0 = 0 \}\),

- \(m\) is the adjacency matrix of a **weighted directed graph**: \(V_i \xrightarrow{m_{ij}} V_j\).

**Example:**

![Diagram](image.png)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>(V_1)</th>
<th>(V_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
<td>+(\infty)</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>(V_1)</td>
<td>-1</td>
<td>+(\infty)</td>
<td>+(\infty)</td>
</tr>
<tr>
<td>(V_2)</td>
<td>-1</td>
<td>1</td>
<td>+(\infty)</td>
</tr>
</tbody>
</table>
The total order on $\mathbb{I}$ is extended to $\mathbb{I}^\defeq \mathbb{I} \cup \{+\infty\}$.

The total order on $\mathbb{I}$ is extended to a partial order on $\mathcal{D}^\#$:

- $m \sqsubseteq^\# n$ defines $\iff \forall i, j, m_{ij} \leq n_{ij}$ point-wise partial order
- $[m \sqcap^\# n]_{ij} \defeq \min(m_{ij}, n_{ij})$ greatest lower bound
- $[m \sqcup^\# n]_{ij} \defeq \max(m_{ij}, n_{ij})$ least upper bound
- $[\top^\#]_{ij} \defeq +\infty$ greatest element

However:

- $m \sqsubseteq^\# n \implies \gamma(m) \subseteq \gamma(n)$ but not the converse,
- $m = n \implies \gamma(m) = \gamma(n)$ but not the converse: $\gamma$ is not injective!

$\implies$ we introduce a normal form.
Normal Form

**Idea:** Derive implicit constraints by summing weights on adjacent arcs:

\[
\begin{aligned}
V_1 - V_2 &\leq 3 \\
V_2 - V_3 &\leq -1 \\
V_1 - V_3 &\leq 4 \\
\end{aligned}
\]

\[
\begin{aligned}
V_3 &\rightarrow V_2 & -1 \\
V_2 &\rightarrow V_1 & 3 \\
V_2 &\rightarrow V_3 & 3 \\
\end{aligned}
\]

\[
\begin{aligned}
V_3 &\rightarrow V_1 & 2 \\
V_1 &\rightarrow V_3 & 4 \\
V_1 &\rightarrow V_2 & 3 \\
\end{aligned}
\]

**Shortest-Path Closure** \( m^* \): Floyd–Warshall algorithm:

\[
\begin{aligned}
m_{ij}^* &\overset{\text{def}}{=} m_{ij}^{n+1} \\
m_{ij}^0 &\overset{\text{def}}{=} m_{ij} \\
m_{ij}^{k+1} &\overset{\text{def}}{=} \min(m_{ij}^k, m_{ik}^k + m_{kj}^k) \quad \text{if } 0 \leq k \leq n \\
\end{aligned}
\]

- derives all implicit constraints in cubic time,
- gives a normal form when \( \gamma(m) \neq \emptyset \): \( m^* = \inf_{n \in \mathbb{N}} \{ n \mid \gamma(n) = \gamma(m) \} \),
- enables emptiness testing: \( \gamma(m) = \emptyset \iff \exists i, m_{ii}^* < 0 \),
- enables inclusion testing: \( \gamma(m) \subseteq \gamma(n) \iff m^* \sqsubseteq n^* \), etc.
Operator Example: Abstract Union

The union of two zones is not always a zone:

\[ \gamma(m) \cup \gamma(n) \subseteq \gamma(m \cup^\# n) \]

But it may not output the smallest zone encompassing two zones... because of implicit constraints.

**Solution:** Define \( m \cup^\# n \stackrel{\text{def}}{=} m^* \cup^\# n^* \):

- always the best abstraction: \( \gamma(m \cup^\# n) = \inf \subseteq \{ \gamma(o) \mid \gamma(m), \gamma(n) \subseteq \gamma(o) \} \)
- \( m \cup^\# n \) is already closed: \( (m \cup^\# n)^* = m \cup^\# n \)

**Note:** The intersection \( \cap^\# \) behaves differently (dually).
We propose several operators with varying cost versus precision trade-offs.

**Exact Assignments:** Only for \( X \leftarrow Y + [a, b], \ X \leftarrow X + [a, b], \) or \( X \leftarrow [a, b]. \)

\[
e.g. \quad \left[ \{ V_{j_0} \leftarrow V_{i_0} + [a, b] \} \#(m) \right]_{ij} \overset{\text{def}}{=} \begin{cases} -a & \text{if } i = j_0 \text{ and } j = i_0, \\ b & \text{if } i = i_0 \text{ and } j = j_0, \\ +\infty & \text{otherwise if } i = j_0 \text{ or } j = j_0, \\ m_{i,j}^* & \text{otherwise.} \end{cases}
\]

**Interval and Polyhedra Based Assignments**

We can reuse existing transfer functions from other abstract domains using:

- **exact** conversion operators: intervals → zones → polyhedra,
- **best** conversion operators: polyhedra → zones → intervals. (using \( * \))

\[
e.g. \quad \begin{array}{l}
\bullet \text{ best abstract assignment for linear expressions using polyhedra}, \\
\bullet \text{ fast assignment of arbitrary expressions using intervals.}
\end{array}
\]
Operator Example: Abstract Assignment

**Problem:** for many usual assignments, e.g., $X \leftarrow Y + Z$:

- there is no exact abstraction,
- the polyhedron-based assignment is too costly, (exponential cost)
- the interval-based assignment is very imprecise. (not relational enough)

$\implies$ we introduce an operator with intermediate cost versus precision.

**Interval Linear Form Assignments:**

\[
V_j \leftarrow [a_0, b_0] + \sum_{k \neq i} ([a_k, b_k] \times V_k)
\]

**For each** $i$, derive new bounds on $V_j - V_i$ by evaluating:

\[
[a_0, b_0] + \sum_{k \neq i} ([a_k, b_k] \times \pi_k(X^\#)) + ([a_i - 1, b_i - 1] \times \pi_i(X^\#))
\]

using the **interval** operators $+$, $\times$, and the interval projections $\pi_k$ of variables $V_k$.

$\implies$ we can **infer** relational invariants for a **linear cost**.

Not optimal because we do not use the relational information in the zone.
Operator Example: Widening

The zone abstract domain has infinite strictly increasing chains!

We need a **widening** $\nabla$ to compute fixpoints in finite time:

\[
\begin{align*}
X_0 & \overset{\text{def}}{=} Y_0 \\
X_i & \overset{\text{def}}{=} X_i \nabla Y_{i+1}
\end{align*}
\]

should converge in **finite time** towards an over-approximation of $\bigcup_i \gamma(Y_i)$

**Example Widening:** Point-wise standard interval widening:

\[
(m \nabla n)_{ij} \overset{\text{def}}{=} \begin{cases} 
  m_{ij} & \text{if } m_{ij} \geq n_{ij} \\
  +\infty & \text{otherwise}
\end{cases}
\]

Unstable constraints are simply thrown away.

**Notes:**

- Any interval widening can be extended point-wisely.
- $X_i \nabla Y_i$ may diverge! Bad interaction between $\ast$ and $\nabla$. 
The Octagon Abstract Domain

Octagons extend zones to invariants of the form:

$$\bigwedge_{i,j} \left( \pm V_i \pm V_j \leq c_{ij} \right) \quad I \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$$

- It is strictly more expressive than the zone domain.
- It has the same asymptotic cost: quadratic in memory and cubic in time.
- It is sufficient to analyse our first example!

The main difficulty is to adapt the normal form algorithm.

All our exactness, best abstraction results derive from the new normal form.
The Zone Congruence Abstract Domain

Zone congruences correspond to invariants of the form:

\[ \bigwedge_{ij} (V_i \equiv V_j + b_{ij}[c_{ij}]) \quad I \in \{ \mathbb{Z}, \mathbb{Q} \} \]

The main difficulty is, again, to adapt the normal form algorithm.

We use a technique similar to Floyd–Warshall’s algorithm in dioid algebras.
Symbolic Manipulation
Core Principles

Idea: Replace expressions with nicer ones.

Suppose that $\forall \rho \in \gamma(\mathcal{X}^\#)$, $\llbracket e \rrbracket(\rho) \subseteq \llbracket e' \rrbracket(\rho)$, then:

$$(\llbracket V \leftarrow e \rrbracket \circ \gamma)(\mathcal{X}^\#) \subseteq (\gamma \circ \llbracket V \leftarrow e' \rrbracket^\#)(\mathcal{X}^\#)$$

$\implies$ we can safely use $\llbracket V \leftarrow e' \rrbracket^\#(\mathcal{X}^\#)$ in place of $\llbracket V \leftarrow e \rrbracket^\#(\mathcal{X}^\#)$.

The same holds for tests.

Example Application:

If $X \in [0, 1]$ in $\mathcal{X}^\#$: we replace $\llbracket V \leftarrow X \times Y \rrbracket^\#(\mathcal{X}^\#)$ with $\llbracket V \leftarrow [0, 1] \times Y \rrbracket^\#(\mathcal{X}^\#)$.

Useful because our abstraction of non-linear assignments is imprecise.

Note: Interactions between numerical abstract values $\mathcal{X}^\#$ and expression transformations. ($\neq$ performing a static program transformation before the analysis)
Linearisation

**Goal:** Put arbitrary expressions to the form \([a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)\).

Useful when we have interval linear form assignment operators. (zones, etc.)

**Interval Linear Form Manipulations:** \(\cdot, \circ, \partial, m, \iota\)

Resemble a vector space structure.

- \([a_0, b_0] + \sum_k [a_k, b_k] \times V_k = ([a_0', b_0'] + \sum_k [a_k', b_k'] \times V_k)
  \begin{equation}
  \overset{\text{def}}{=} ([a_0, b_0] + [a_0', b_0']) + \sum_k ([a_k, b_k] + [a_k', b_k']) \times V_k
  \end{equation}

- \([a, b] \circ ([a_0', b_0'] + \sum_k [a_k', b_k'] \times V_k) = ([a, b] \times [a_0', b_0']) + \sum_k ([a, b] \times [a_k', b_k']) \times V_k

- \(\iota([a_0, b_0] + \sum_k [a_k, b_k] \times V_k, \mathcal{X}^\#) = [a_0, b_0] + \sum_k ([a_k, b_k] \times \pi_k(\mathcal{X}^\#))
  \text{(on-the-fly intervalisation)}\)
Linearisation (continued)

Linearising an expression: \(|e|\) defined by structural induction:

- \(|V_i\)(\(X^\#\)) \(\overset{\text{def}}{=} [1, 1] \times V_i\)
- \(|e_1+e_2\)(\(X^\#\)) \(\overset{\text{def}}{=} (|e_1|)(\(X^\#\)), (|e_2|)(\(X^\#\))\)
- \(|e_1 \times e_2\)(\(X^\#\)) \(\overset{\text{def}}{=} [a, b] \, |e_2|)(\(X^\#\)) \text{ when } (|e_1|)(\(X^\#\)) = [a, b]\)
- \(|e_1 \times e_2\)(\(X^\#\)) \(\overset{\text{def}}{=} \iota((|e_1|)(\(X^\#\)), \(X^\#\), \(X^\#\)) \, |e_2|)(\(X^\#\)) \text{ or } \iota((|e_2|)(\(X^\#\)), \(X^\#\), \(X^\#\)) \, |e_1|)(\(X^\#\))\)

Notes:

- ♦ Non-linear multiplication: we must choose whether to intervalise \(e_1\) or \(e_2\).
  example: intervalise the expression with smallest bounds
  \(X \in [0, 1], Y \in [-10, 10] \implies (|X \times Y|)(\(X^\#\)) = [0, 1] \times Y\)

- ♦ Linearisation provides simplification for free: \(|(X + Y) - X|)(\(X^\#\)) = Y\).
  If \(X, Y \in [0, 1]\), interval arithmetics gives \([|(X + Y) - X|^\# = [-1, 2]\) but \([|Y|^\# = [0, 1].\)
Symbolic Constant Propagation


Example: \( X \leftarrow Y + Z; U \leftarrow X - Z \)

- \( \{ U \leftarrow X - Z \}^\# \) is replaced with \( \{ U \leftarrow (Y + Z) - Z \}^\# \),
- which is linearised into \( \{ U \leftarrow Y \}^\# \).

Technique: \( \mathcal{X}^\# \in \mathcal{D}^\# \) is enriched with a map \( S^\# \in (\mathcal{V} \rightarrow \mathcal{E}) \).

- Abstract elements \( < \mathcal{X}^\#, S^\# > \) now represent:
  \[ \gamma < \mathcal{X}^\#, S^\# > \overset{\text{def}}{=} \{ \rho \in \gamma(\mathcal{X}^\#) \mid \forall i, \rho(V_i) \in [S^\#(V_i)](\rho) \}. \]

- Abstract assignments \( \{ X \leftarrow e \}^\# < \mathcal{X}^\#, S^\# > \)
  - **propagate** \( S^\# \) into \( e \) to get \( e' \) and evaluate \( \mathcal{X}'^\# \overset{\text{def}}{=} \{ X \leftarrow e' \}^\#(\mathcal{X}^\#) \),
  - kill information on \( X \) in \( S^\# \), then add \( X = e \).

Note: We must **choose** how far to propagate.
Floating-Point Number Abstractions
IEEE 754-1985 Floating-Point Numbers:

We consider the IEEE 754-1985 norm because:

♦ it is widely implemented in today’s hardware (Intel, Motorola),
♦ it is supported by the C language (and many others).

Example: 32-bit “single precision” float numbers $\mathbb{F}$

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent $e$</th>
<th>Fraction $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$e_8 \cdots e_1$</td>
<td>$b_1 \cdots b_{23}$</td>
</tr>
</tbody>
</table>

The set $\mathbb{F}$ of floats is composed of:

♦ **normalised** numbers: $(-1)^s \times 2^{e-127} \times 1.b_1 \cdots b_{23}$
  $(1 \leq e \leq 254)$

♦ **denormalised** numbers: $(-1)^s \times 2^{-126} \times 0.b_1 \cdots b_{23}$
  $(e = 0, b \neq 0)$

♦ **signed zeros**: $+0$ and $-0$

♦ **infinities and error codes**: $+\infty$, $-\infty$, $NaN$
  $(if \ e = 255)$
Floating-Point Expressions $\mathcal{E}_f$:

$$
\mathcal{E}_f ::= \begin{array}{l}
[a, b] \quad \text{interval } a, b \in \mathbb{F} \\
X \quad \text{variable } X \in \mathcal{V} \\
\oplus \mathcal{E}_f \quad \text{unary operator} \\
\mathcal{E}_f \odot \mathcal{E}_f \quad \text{binary operators } \odot \in \{\oplus, \otimes, \ldots\}
\end{array}
$$

Floating-Point Arithmetics:

Differences between floating-point and $\mathbb{Q}$, $\mathbb{R}$ arithmetics:

- **rounding** to a representable float occurs, several types of rounding: **towards** $+\infty$, $-\infty$, 0 or **to nearest**.

- **overflow**: large numbers, division by 0 generate $+\infty$ or $-\infty$, 

- **underflow**: small numbers round to $+0$ or $-0$, 

- **invalid operations**: $0/0$, $(+\infty) + (-\infty)$, etc. generate $NaN$. 

PhD. Defense - Weakly Relational Numerical Abstract Domains - Antoine Miné
Chosen Floating-Point Semantics

Restrict to programs that use $\mathbb{F}$ as “approximated reals”:

♦ Rounding and underflow are benign,
  but we must consider all rounding directions!

♦ Overflow and invalid operations result in a run-time error $\Omega$.

♦ Error-free computations live in $\mathbb{F}' \simeq \mathbb{F} \cap \mathbb{R}$, assimilated to a finite subset of $\mathbb{R}$.

Partial Definition of $\llbracket e \rrbracket_f$: (with rounding towards $+\infty$)

- $\llbracket e_1 \oplus e_2 \rrbracket_f(\rho) \overset{\text{def}}{=} \{ R(v_1 + v_2) \mid v_1 \in \llbracket e_1 \rrbracket_f(\rho), v_2 \in \llbracket e_2 \rrbracket_f(\rho) \}$,
- etc.

- $R(x) \overset{\text{def}}{=} \begin{cases} \Omega & \text{if } x = \Omega \text{ or } x > 2^{127}(2 - 2^{-23}) \\ \min \{ y \in \mathbb{F}' \mid y \geq x \} & \text{otherwise} \end{cases}$
- etc.
Difficulties in Adapting Relational Domains

♦ The interval domain is easy to adapt. We simply round lower bounds toward $-\infty$ and upper bounds toward $+\infty$.

♦ Relational domains cannot manipulate floating-point expressions. Such domains require properties of $\mathbb{Q}$, $\mathbb{R}$ not true in floating-point arithmetics!

\[
\begin{align*}
(X - Y \leq c) \land (Y - Z \leq d) &\implies (X - Z \leq c + d) \\
(X \ominus Y \leq c) \land (Y \ominus Z \leq d) &\nleftrightarrow (X \ominus Z \leq c \oplus d) \\
(10^{22} \oplus 1.000000019 \cdot 10^{38}) \oplus (10^{22} \ominus 1.000000019 \cdot 10^{38}) &= 0
\end{align*}
\]

Solution:

• $[e]_f$ is abstracted as a linear interval form on $\mathbb{Q}$.

• Invariant semantics will be expressed using $\mathbb{Q}$, $+$, $-$, \ldots not $\mathbb{F}'$, $\oplus$, $\ominus$.

$\implies$ We keep the same abstract domains and operators as before.
Floating-Point Linearisation

Rounding Error on Linear Forms: Its magnitude is the maximum of:

♦ a relative error \( \varepsilon \) of amplitude \( 2^{-23} \), expressed as a linear form:

\[
\varepsilon([a, b] + \sum_i [a_i, b_i] \times V_i) \overset{\text{def}}{=} \max(|a|, |b|) \times [-2^{-23}, 2^{-23}] + \sum_i (\max(|a_i|, |b_i|) \times [-2^{-23}, 2^{-23}]) \times V_i
\]

(normalised numbers)

♦ an absolute error \( \omega \) \overset{\text{def}}{=} [-2^{-159}, 2^{-159}] \) (denormalised numbers).

\( \Rightarrow \) We sum these two causes of rounding.

Linearisation \((e) f\):

- \( ([e_1 \oplus e_2]) f(X^\#) \overset{\text{def}}{=} ([e_1] f(X^\#) \oplus [e_2] f(X^\#)) \cdot \varepsilon([e_1] f(X^\#)) \cdot \varepsilon([e_2] f(X^\#)) \cdot \omega \)
- \( ([a, b] \otimes e_2) f(X^\#) \overset{\text{def}}{=} ([a, b] \otimes [e_2] f(X^\#)) \cdot ([a, b] \otimes \varepsilon([e_2] f(X^\#))) \cdot \omega \)
- etc.
Application of Floating-Point Linearisation

Abstract Assignment: \( V \leftarrow e \)

We first evaluate \( e \) in the floating-point interval domain.

- If there is no run-time error \( \Omega \) detected, then
  \[
  \forall \rho \in \gamma(\mathcal{X}^\#),\; \llbracket e \rrbracket_f(\rho) \subseteq \llbracket (e) f(\mathcal{X}^\#) \rrbracket(\rho)
  \]
  and we can feed \( \{ V \leftarrow (e) f(\mathcal{X}^\#) \}^\# \) to an abstract domain in \( \mathcal{Q} \).

- If \( \Omega \) is detected, we can still fall back to the interval domain.

Example: \[
Z \leftarrow X \ominus (0.25 \otimes X)
\]
is linearised as
\[
Z \leftarrow ([0.749 \cdots , 0.750 \cdots ] \times X) + (2.35 \cdots 10^{-38} \times [-1, 1])
\]
- Allows simplification even in the interval domain.
  e.g., if \( X \in [-1, 1] \), we get \( |Z| \leq 0.750 \cdots \) instead of \( |Z| \leq 1.25 \cdots \)
- Allows using a relational abstract domain. (zone, etc.)
Floating-Point Zones

We are now sound, but not very efficient: abstract operations are expressed in $\mathbb{Q}$. This requires costly arbitrary precision exact rational packages!

**Solution:** Perform all abstract computations in $\mathbb{F}$:

- **linearisation:** use sound floating-point interval arithmetics,
- **zone domain:** upper bounds computation are rounded towards $+\infty$.

We lose some precision.
We gain much speed.

**Note:** Sound algorithms in $\mathbb{F}$ are much harder to provide for polyhedra!
Floating-Point Abstractions

To sum up, the following sound approximations are made:

① **linearisation**: rounding errors are treated as non-deterministic,
② **linearisation**: non-linear computations are “intervalised”,
③ **abstract domain**: limits the expressiveness,
④ **abstract operators**,
⑤ **implementation in F**: extra rounding errors.

Due to ① and ⑤, our best abstraction results no longer hold!

Despite unpredictable ⑤, abstract computations are stable in many cases:

- when concrete computations are naturally **contracting**, e.g., \( X ← 0.5X + [−1, 1] \),
- when concrete computations have explicit **limiters**,
- specific **widenings** and **narrowings** can help.
Real-Life Application Within Astrée
Presentation of Astrée

Astrée:

♦ Static analyser developed at the ENS.
♦ Checks for run-time errors in reactive C code. (integer and float overflows, etc.)
♦ Aimed at proving automatically the correctness: 0 alarm goal.

Analysed Code Features:

A real-life example:

• primary flight control software for the Airbus A340 fly-by-wire system,
• 70,000 lines of C,
• 10,000 global variables, 5,000 of which are 32-bit floating-point,
• one very large loop executed $3.6 \cdot 10^6$ times.
Octagon Packing

Problem: There are too many variables (10,000) even for the octagon domain!

Solution: Do not relate all variables together.

♦ Define static packs of a few variables only.
♦ One octagon per pack, no inter-pack relationality.

Automatic Packing: Using simple syntactic criteria.

<table>
<thead>
<tr>
<th># lines</th>
<th># variables</th>
<th># packs</th>
<th>pack size</th>
</tr>
</thead>
<tbody>
<tr>
<td>370</td>
<td>100</td>
<td>20</td>
<td>3.6</td>
</tr>
<tr>
<td>9,500</td>
<td>1,400</td>
<td>200</td>
<td>3.1</td>
</tr>
<tr>
<td>70,000</td>
<td>14,000</td>
<td>2,470</td>
<td>3.5</td>
</tr>
<tr>
<td>226,000</td>
<td>47,500</td>
<td>7,429</td>
<td>3.5</td>
</tr>
<tr>
<td>400,000</td>
<td>82,000</td>
<td>12,964</td>
<td>3.3</td>
</tr>
</tbody>
</table>

⇒ Linear increase in cost: the method is scalable.
Analysis Results

Astrée includes:

- floating-point octagons using floating-point linearisation,
- symbolic propagation in the interval domain,
- other domains working in $\mathbb{R}$, supplied with linearised floating-point expressions.

Analysis Comparison:  
AMD Opteron 248, mono-processor

<table>
<thead>
<tr>
<th># lines</th>
<th>without symbolic</th>
<th></th>
<th>without octagon</th>
<th></th>
<th>with everything</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>memory</td>
<td>alarms</td>
<td>time</td>
<td>memory</td>
<td>alarms</td>
</tr>
<tr>
<td>370</td>
<td>1.8s</td>
<td>16 MB</td>
<td>0</td>
<td>1.7s</td>
<td>14 MB</td>
<td>0</td>
</tr>
<tr>
<td>9 500</td>
<td>90s</td>
<td>81 MB</td>
<td>8</td>
<td>75s</td>
<td>75 MB</td>
<td>8</td>
</tr>
<tr>
<td>70 000</td>
<td>2h 40mn</td>
<td>559 MB</td>
<td>391</td>
<td>3h 17mn</td>
<td>537 MB</td>
<td>58</td>
</tr>
<tr>
<td>226 000</td>
<td>11h 16mn</td>
<td>1.3 GB</td>
<td>141</td>
<td>7h 8mn</td>
<td>1.0 GB</td>
<td>165</td>
</tr>
<tr>
<td>400 000</td>
<td>22h 8mn</td>
<td>2.2 GB</td>
<td>282</td>
<td>20h 31mn</td>
<td>1.7 GB</td>
<td>804</td>
</tr>
</tbody>
</table>

⇒ Our work is instrumental in proving the code correctness!
Conclusion
Work Summary

To sum up we proposed:

♦ New relational abstract domains between intervals and polyhedra.

  Provides new theoretical results. (properties of closure)
  Design and proofs of soundness, exactness, best precision of abstract operators.

♦ Generic techniques for the local enhancement of domains:
  Linearisation, symbolic constant propagation.

  Avoid the need for more expressive domains.

♦ Adaptation to floating-point arithmetics.

  First relational domains to relate floating-point variable values.

♦ Integration within the Astrée analyser.

  Motivated new researches. (abstract operators, packing, etc.)
  Provided experimental results on real-life examples.
## Abstract Domains Comparison

<table>
<thead>
<tr>
<th>Domain</th>
<th>Invariants</th>
<th>Cost</th>
<th>Floating-Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intervals</td>
<td>$X \in [a, b]$</td>
<td>$\mathcal{O}(n)$</td>
<td>yes</td>
</tr>
<tr>
<td>Zones</td>
<td>$X - Y \leq c$</td>
<td>$\mathcal{O}(n^2, n^3)$</td>
<td>yes</td>
</tr>
<tr>
<td>Octagons</td>
<td>$\pm X \pm Y \leq c$</td>
<td>$\mathcal{O}(n^2, n^3)$</td>
<td>yes</td>
</tr>
<tr>
<td>Zone congruence</td>
<td>$X \equiv Y + a [b]$</td>
<td>$\mathcal{O}(n^2, n^3)$</td>
<td>no</td>
</tr>
<tr>
<td>Symbolic</td>
<td>$X = \mathcal{E}$</td>
<td>$\mathcal{O}(n)$</td>
<td>yes</td>
</tr>
<tr>
<td>Polyhedra</td>
<td>$\sum_i \alpha_i X_i \leq \beta$</td>
<td>$\mathcal{O}(e^n)$</td>
<td>no</td>
</tr>
</tbody>
</table>

The ability to easily implement floating-point versions is crucial.
Future Work

♦ Extent the *spectrum choice for cost vs. precision trade-offs*:

- Define new abstract domains.  
  (e.g., between octagons and polyhedra; Octahedra, TVPI)
- Define alternate abstract operators.  
  (fine-grain control, widenings)
- Local refinement techniques, non-homogeneous precision  
  (extend packing)
- Theoretical results on linearisation and symbolic propagation techniques.  
  (precision guarantees)

♦ Consider new numerical properties, *adapted to*:

- Complex numerical algorithms.  
  (finite elements methods)
- Non-numerical properties parametrised by a numerical domain.  
  (e.g., non-uniform pointer analysis)
- Parametric predicate abstractions.  
  (complex functional properties, e.g., sorting algorithms)
Thank you for your attention!
Octagon Analysis Example (1)

Absolute Value Computation:

\[
X \leftarrow [-100, 100] \\
① Y \leftarrow X \\
② \text{if } Y \leq 0 \{ ③ Y \leftarrow -Y ④ \} \text{ else } \{ ⑤ \} \\
⑥ \text{if } Y \leq 69 \{ ⑦ \ldots \} \\
\]

The octagon domain can prove that, at ⑦, \(-69 \leq X \leq 69\).
Rate Limiter:

\[
\begin{align*}
Y &\leftarrow 0 \\
\text{while } & \begin{cases} \text{random()} \{ \\
X &\leftarrow [-128, 128] \\
D &\leftarrow [0, 16] \\
S &\leftarrow Y \\
R &\leftarrow X - S \\
Y &\leftarrow X \\
\text{if } R \leq -D \{ Y &\leftarrow S - D \} \text{ else} \\
\text{if } D \leq R \{ Y &\leftarrow S + D \} \\
\} \\
\end{cases}
\end{align*}
\]

The octagon domain can prove that \(|Y| \leq M\) is \textbf{stable} at ① for any \(M \geq 144\).

In fact, we have \(Y \in [-128, 128]\)…

\textbf{Note: } The interval domain \textbf{cannot} prove any bound to be stable.
Interaction Between Closure and Widening

\[ \begin{align*}
X & \leftarrow 0 \\
Y & \leftarrow [-1, 1] \\
\text{while random()} & \{ \\
& \quad \text{if } X = Y \{ \\
& \quad \quad \text{if random()} \{ Y \leftarrow X + [-1, 1] \} \\
& \quad \quad \text{else} \quad \{ X \leftarrow Y + [-1, 1] \} \\
& \quad \} \\
& \} \\
\end{align*} \]

Non-Terminating Analysis:

Using \( m_{i+1} \overset{\text{def}}{=} (m_i^*) \bigtriangleup n_i \)

\[ \begin{align*}
\text{m}_0 & \overset{\text{def}}{=} (1, 1, 0) \\
\text{n}_i & \overset{\text{def}}{=} (i+1, i+1, i+1) \\
\text{m}_{2i} & = (2i+1, 2i+1, 2i) \\
\text{m}_{2i+1} & = (2i+1, 2i+2, 2i+2) \\
\end{align*} \]