Workshop on Numerical & Symbolic Abstract Domains

Weakly Relational Numerical Abstract Domains

Theory and Application

Antoine Miné

École Normale Supérieure, Paris

January 21st 2005
What Are Numerical Abstract Domains For?

In an **Abstract Interpretation** based analyser they can:
- 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 4th 1996 $\rightarrow$ $500$ M loss)
- Check for out-of-bound array or pointer arithmetics.
  (50% of Unix vulnerabilities according to CERT)
- Optimisation, debugging information inference.
- Parameter to non-numerical analyses.
  (pointer analyses [Venet], parametric predicate abstractions [Cousot], etc.)
Traditional Numerical Domains

Non-Relational Domains:

Constant Propagation
\[ X_i = c_i \]
[Kildall73]

Signs
\[ X_i \geq 0, \; X_i \leq 0 \]
[CC76]

Intervals
\[ X_i \in [a_i, b_i] \]
[CC76]

Simple Congruences
\[ X_i \equiv a_i \; [b_i] \]
[Granger89]

Interval Congruences
\[ X_i \in \alpha_i [a_i, b_i] \]
[Masdupuy93]

Power Analysis
\[ X_i \in \alpha_i^{a_i \mathbb{Z} + b_i}, \alpha_i^{[a_i, b_i]}, \text{etc.} \]
[Mastroeni01]
Traditional Numerical Domains

Relational Domains:

Linear Equalities
\[ \sum_i \alpha_{ij} X_i = \beta_j \]
[Karr76]

Linear Congruences
\[ \sum_i \alpha_{ij} X_i \equiv \beta_j [\gamma_j] \]
[Granger91]

Trapezoidal Congruences
\[ X_i = \sum_j \lambda_j \alpha_{ij} + \beta_j \]
[Masdupuy92]

Polyhedra
\[ \sum_i \alpha_{ij} X_i \leq \beta_j \]
[CH78]

Ellipsoids
\[ \alpha X^2 + \beta Y^2 + \gamma XY \leq \delta \]
[Feret04]

Varieties
\[ P_i(\vec{X}) = 0, \ P_i \in \mathbb{R}[\mathcal{V}] \]
[R-CK04]
Recent Issues In Numerical Domains

- Granularity in the cost vs. precision trade-off is **too coarse**.

  Few domains can infer **bounds**:
  - the interval domain is too **imprecise** (non relational)
  - the polyhedron domain is too **costly**
    (unbounded in theory, exponential in practice)

  \[\Rightarrow\] we can
  - define some **new** relational domains
  - **tweak** the cost vs. precision trade-off of existing domains

- Relational domains are **not sound** on machine-integers & floating-point numbers!
  ([Simon], [Goubault and Putot] may talk about this...)
The Need for Relational Domains

Loop Invariant Inference

Finding a non-relational property may require the inference of a relational loop invariant.

Example:

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

The interval domain will only find V ≥ 0 and I = -1 at •.

To prove that V ≤ 11, we need to prove a relational loop invariant at •: V + I ≤ 10.
The Need for Relational Domains

Other applications of relationality:

- precise analysis of assignments and tests involving several variables, (we will see examples shortly...)

- analysis of programs with *symbolic* parameters,

- *modular* analysis of procedures, classes, modules, etc.

- inference of *non-uniform* non-numerical invariants. (e.g., non-uniform pointer aliasing analysis [Venet])
Overview

- Formal framework.

- New **numerical abstract domains**: zones and octagons.

- Static **variable packing** technique to cut costs.

- **Symbolic manipulation** techniques to improve precision.

- Adaptation to **floating-point** semantics.

- Application within the **Astrée** analyser and **experimental** results.

Form theory to application... and backwards.
Formal Framework
Language Syntax

We first consider an **idealised** language:

- 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

\[
\mathcal{I} ::= \begin{align*}
X & \leftarrow E & \text{assignment to } X \in \mathcal{V} \\
E & \bowtie 0 \ ? & \text{test } \bowtie \in \{ =, \leq, \ldots \}
\end{align*}
\]

### Expressions

\[
\mathcal{E} ::= \begin{align*}
[a, b] & \text{ interval } a \in \mathbb{I} \cup \{ -\infty \}, \ b \in \mathbb{I} \cup \{ +\infty \}, \\
X & \text{ variable } X \in \mathcal{V} \\
-\ E & \text{ unary operator} \\
E \odot E & \text{ binary operators } \odot \in \{ +, \times, \ldots \}
\end{align*}
\]

**Notes:**

- \([a, b]\) models a **non-deterministic** choice within an interval,
- adaptation to machine-integers and floating-point variables will come later,
- other language features are orthogonal.
Concrete Semantics

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

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

$E$ maps **environments** to **sets** of numerical values:

\[
\llbracket [a, b] \rrbracket(\rho) \overset{\text{def}}{=} \{ c \in \mathbb{I} | a \leq c \leq b \} \\
\llbracket X \rrbracket(\rho) \overset{\text{def}}{=} \{ \rho(X) \} \\
\llbracket e_1 + e_2 \rrbracket(\rho) \overset{\text{def}}{=} \{ v_1 + v_2 | v_1 \in \llbracket e_1 \rrbracket(\rho), v_2 \in \llbracket e_2 \rrbracket(\rho) \} \\
\llbracket e_1 / e_2 \rrbracket(\rho) \overset{\text{def}}{=} \{ v_1 / v_2 | v_1 \in \llbracket e_1 \rrbracket(\rho), v_2 \in \llbracket e_2 \rrbracket(\rho) \setminus \{0\} \} \\
\]

etc.

There is no error state: run-time errors **halt** the program and are not propagated.
Concrete Semantics

Instruction Semantics: \( \{ I \} : \mathcal{P}(V \rightarrow I) \rightarrow \mathcal{P}(V \rightarrow I) \)

A transfer function defines a relation between environments:

- Assignments:
  \[
  \{ X \leftarrow e \}(R) \overset{\text{def}}{=} \{ \rho[ X \rightarrow v ] \mid \rho \in R, \ v \in \llbracket e \rrbracket(\rho) \}
  \]

- Tests: filter environments
  \[
  \{ e \triangleright 0 ? \}(R) \overset{\text{def}}{=} \{ \rho \in R \mid \exists v \in \llbracket e \rrbracket(\rho) \text{ such that } v \triangleright 0 \}
  \]

- Backwards assignments:
  \[
  \{ X \rightarrow e \}(R) \overset{\text{def}}{=} \{ \rho \mid \exists v \in \llbracket e \rrbracket(\rho), \ \rho[ X \leftarrow v ] \in R \}
  \]

useful to
- refine abstract semantics by backwards / forward iterations,
- perform abstract debugging.
Concrete Semantics

Given a control-flow graph \((L, e, I)\):  
\[
\begin{array}{l}
L & \text{program points} \\
e & \in L & \text{entry point} \\
I & \subseteq L \times I \times L & \text{arcs}
\end{array}
\]

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

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

that gathers all possible environments at each program point.

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

\(\implies\) we will compute **sound over-approximations** of the \(\mathcal{X}_l\)\ldots
Abstract Domains: Formal Definition

We will work in the Abstract Interpretation framework, a general theory of sound approximations of semantics [Cousot78].

Numerical Abstract Domain:

- **computer-representable** set $\mathcal{D}^\#$ of abstract values, together with:
  - a *concretisation*: $\gamma: \mathcal{D}^\# \rightarrow \mathcal{P}(\mathcal{V} \rightarrow \mathbb{I})$,
  - a *partial order*: $\sqsubseteq^\#, \bot^# , \top^#$,
  - sound, effective abstract transfer functions $\{ I \}^\#$: $(\{ I \} \circ \gamma)(\mathcal{X}^\#) \subseteq (\gamma \circ \{ I \}^\#)(\mathcal{X}^\#)$,
  - 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.
The Zone Abstract Domain
The Zone Abstract Domain

Less expressive but simpler than the octagon domain.

Zones enrich intervals with invariants of the form:

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

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 \(\mathbb{I} \cup \{+\infty\}\):
  - \(m_{ij} \neq +\infty\) 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: \(-V_i \leq m_{i0}, V_j \leq m_{0j}\),

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

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

**Example:**

![Diagram](image.png)

*NSAD'05 - Weakly Relational Numerical Abstract Domains - Antoine Miné*
Order Structure

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 $\mathcal{D}^\#$:

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

However:

- $\mathbf{m} \sqsubseteq^\mathbf{#} \mathbf{n} \implies \gamma(\mathbf{m}) \subseteq \gamma(\mathbf{n})$ but not the converse,
- $\mathbf{m} = \mathbf{n} \implies \gamma(\mathbf{m}) = \gamma(\mathbf{n})$ 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:

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

\[
\begin{array}{c}
\text{e.g. } \\
\begin{mymatrix}
V_1 & V_2 & V_3 \\
V_3 & -1 & 3 & 4 \\
V_2 & 1 & -1 & \end{mymatrix} \Rightarrow \\
\begin{mymatrix}
V_1 & V_2 & V_3 \\
V_3 & -1 & 3 & 2 \\
V_2 & 1 & -1 & \end{mymatrix}
\end{array}
\]

\[
\begin{align*}
V_1 - V_2 &\leq 3 \\
V_2 - V_3 &\leq -1 \\
V_1 - V_3 &\leq 2
\end{align*}
\]

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

\[
\begin{align*}
\begin{cases}
m_{i,j}^* &\overset{\text{def}}{=} m_{i,j}^{n+1} \\
m_{i,j}^0 &\overset{\text{def}}{=} m_{i,j} \\
m_{i,j}^{k+1} &\overset{\text{def}}{=} \min(m_{i,j}^k, m_{i,k}^k + m_{k,j}^k) & \text{if } 0 \leq k \leq n
\end{cases}
\end{align*}
\]

- derives all implicit constraints in cubic time,
- gives a normal form when \(\gamma(m) \neq \emptyset\): \(m^* = \inf \subseteq \{ n \mid \gamma(n) = \gamma(m) \} \),
- enables emptiness testing: \(\gamma(m) = \emptyset \iff \exists i, m_{i,i}^* < 0\),
- 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:

\[
\begin{align*}
\begin{array}{c}
\text{Zone 1} \\
\text{Zone 2}
\end{array}
\rightarrow
\begin{array}{c}
\text{Union of Zones 1 and 2}
\end{array}
\end{align*}
\]

\(\square\) is a sound counterpart for \(\sqcup\): \(\gamma(m) \cup \gamma(n) \subseteq \gamma(m \sqcup n)\).

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

Solution: Define \(m \sqcup n \triangleq m^* \sqcup n^*\):

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

Note: The intersection \(\sqcap\) behaves differently (dually).
Operator Example: Abstract Assignment

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 $X \leftarrow [a, b]$.

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

**Interval and Polyhedra Based Assignments**

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

- exact conversion operators: intervals $\rightarrow$ zones $\rightarrow$ polyhedra,
- best conversion operators: polyhedra $\rightarrow$ zones $\rightarrow$ intervals. (using $\ast$)

**e.g.**

- best abstract assignment for linear expressions using polyhedra,
- fast assignment of arbitrary expressions using intervals.
**Operator Example: Abstract Assignment**

**Problem:** for many usual assignments, e.g., \( X \leftarrow Y + Z \):
- there is no exact abstraction,
- the interval-based assignment is very imprecise, (not relational enough)
- the polyhedron-based assignment is too costly, (exponential cost)
  (LP as in [Sankaranarayanan et al.] may solve this problem...)

\[ \implies \text{we introduce an operator with intermediate cost versus precision.} \]

**Interval Linear Form Assignments:** \( V_j \leftarrow [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k) \)

For each \( i \), derive new bounds on \( V_j - V_i \) by evaluating:

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

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

\[ \implies \text{we can infer relational invariants for a linear cost.} \]

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

Precision Comparison:

Argument

\[
\begin{align*}
0 & \leq Y \leq 10 \\
0 & \leq Z \leq 10 \\
0 & \leq Y - Z \leq 10
\end{align*}
\]

\[\Downarrow \quad X \leftarrow Y - Z\]

\[
\begin{align*}
-10 & \leq X \leq 10 \\
-20 & \leq X - Y \leq 10 \\
-20 & \leq X - Z \leq 10
\end{align*}
\]

- Interval-based

\[
\begin{align*}
-10 & \leq X \leq 10 \\
-10 & \leq X - Y \leq 0 \\
-10 & \leq X - Z \leq 10
\end{align*}
\]

- Interval linear form based

\[
\begin{align*}
0 & \leq X \leq 10 \\
-10 & \leq X - Y \leq 0 \\
-10 & \leq X - Z \leq 10
\end{align*}
\]

- Polyhedron-based

(best)

Full analysis examples will be presented shortly — within the octagon domain.
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 Widenings:

Point-wise extensions of interval widenings:

- **standard widening**: throw away unstable constraints

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

- **widening with thresholds** $T$ ($T$ is a finite set)

  $$
  (m \nabla n)_{ij} \overset{\text{def}}{=} \begin{cases} 
  m_{ij} & \text{if } m_{ij} \geq n_{ij} \\
  \min \left\{ x \in T \cup \{+\infty\} \mid x \geq n_{ij} \right\} & \text{otherwise}
  \end{cases}
  $$
Operator Example: Widening

**Important Note:**

\[ X_i^{##} \overset{\text{def}}{=} (X_i^{###}) \nabla Y_i^{##+1} \text{ may diverge!} \]

This is because:

- \( \nabla \) termination is enforced by setting coefficients to \( +\infty \)
- \( \ast \) **tightens** \( +\infty \) **coefficients** into finite ones

This is very unlike other operators \( \sqcup^{##}, \lbrack \cdot \rbrack^{##}, \) etc., that **benefit** from closure.

([Sankaranarayanan et al.] avoid refining widened constraints)
([Bagnara et al.] may have another answer. . . )

**Semantical Widening:**

**Open problem:** find a widening independent from the chosen DBM representation.
(c.f., polyhedron widening)
The Octagon Abstract Domain
The Octagon Abstract Domain

Octagons extend zones to invariants of the form:

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

- Strictly **more expressive** than the zone domain.
- Same asymptotic cost: **quadratic** in memory and **cubic** in time.
- Precise enough to analyse our loop example. (and more...)
Octagon Representation

We still use DBMs!

**Idea:** Rewrite octagonal constraints as potential constraints on $\mathcal{V}' \stackrel{\text{def}}{=} \{V'_1, \ldots, V'_{2n}\}$.

- $V'_{2k-1}$ represents $+V_k$
- $V'_{2k}$ represents $-V_k$

<table>
<thead>
<tr>
<th>the constraint</th>
<th>is represented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_i - V_j \leq c$ \hspace{1cm} ($i \neq j$)</td>
<td>$V'<em>{2i-1} - V'</em>{2j-1} \leq c$ and $V'_j - V'_i \leq c$</td>
</tr>
<tr>
<td>$V_i + V_j \leq c$ \hspace{1cm} ($i \neq j$)</td>
<td>$V'<em>{2i-1} - V'</em>{2j} \leq c$ and $V'_j - V'_i \leq c$</td>
</tr>
<tr>
<td>$-V_i - V_j \leq c$ \hspace{1cm} ($i \neq j$)</td>
<td>$V'<em>{2i} - V'</em>{2j-1} \leq c$ and $V'_j - V'_i \leq c$</td>
</tr>
<tr>
<td>$V_i \leq c$</td>
<td>$V'<em>{2i-1} - V'</em>{2i} \leq 2c$</td>
</tr>
<tr>
<td>$V_i \geq c$</td>
<td>$V'<em>i - V'</em>{2i-1} \leq -2c$</td>
</tr>
</tbody>
</table>

**Adapted Concretisation:** of a DBM $m$ of size $2n \times 2n$

$$\gamma(m) \stackrel{\text{def}}{=} \{ (v_1, \ldots, v_n) \mid \forall i, j, v'_j - v'_i \leq m_{ij}, v'_{2i-1} = -v'_{2i} = v_i \}$$
**Octagon Representation**

**Coherence:**

One octagon constraint can have two encodings. We require the two encodings to represent the same constraint:

\[ \forall i, j, m_{ij} = m_{\overline{i} \overline{j}} \text{ where } \overline{i} \overset{\text{def}}{=} \begin{cases} i - 1 & \text{if } i \text{ is even} \\ i + 1 & \text{if } i \text{ is odd} \end{cases} \]

**Octagon Example:**

\[
\begin{align*}
V_1 + V_2 &\leq 3 \\
V_2 - V_1 &\leq 3 \\
V_1 - V_2 &\leq 3 \\
-V_1 - V_2 &\leq -3 \\
2V_2 &\leq 2 \\
-2V_2 &\leq 8
\end{align*}
\]
Adapted Normal Form

The shortest-path closure is not a normal form. We must take into account the implicit constraints $V'_{2i-1} + V'_{2i} = 0$.

**Strongly Closed DBM:** when $\not\in \mathbb{Z}$

- $\forall i, j, k, \ m_{ij} \leq m_{ik} + m_{kj}$ (closed by transitivity)
- $\forall i, j, \ m_{ij} \leq (m_{ii} + m_{jj})/2$ (closed by addition of unary constraints)

**Properties:**

- Each constraint in a strongly closed DBM is saturated.
- There is a unique strongly closed DBM representing a non-empty octagon.
- We can construct complete equality and inclusion tests.
- We can construct best, exact operators.
Adapted Normal Form

Modified Floyd–Warshall Algorithm $m^\bullet$: when $\mathbb{I} \neq \mathbb{Z}$

we define:

$$
\begin{align*}
  m^\bullet & \overset{\text{def}}{=} m^n \\
  m^0 & \overset{\text{def}}{=} m \\
  m^{k+1} & \overset{\text{def}}{=} S(C^{2k+1}(m^k)) \text{ if } 0 \leq k < n
\end{align*}
$$

where:

$$
(S(n))_{ij} \overset{\text{def}}{=} \min(n_{ij}, (n_{i\overline{j}} + n_{\overline{i}j})/2)
$$

$$
(C^k(n))_{ij} \overset{\text{def}}{=} \min(n_{ij}, n_{ik} + n_{kj}, n_{i\overline{k}} + n_{\overline{k}j},
  n_{ik} + n_{k\overline{k}} + n_{\overline{k}j}, n_{i\overline{k}} + n_{k\overline{k}} + n_{kj})
$$

Properties:

- Emptiness test: $\gamma(m) = \emptyset \iff \exists i, m^n_{ii} < 0$.
- If $\gamma(m) \neq \emptyset$, $m^\bullet$ is strongly closed.
- $m^\bullet$ can be computed in cubic time.
- All operators are constructed as in the zone domain, using $\bullet$ instead of $\ast$. 

NSAD’05 - Weakly Relational Numerical Abstract Domains - Antoine Miné
Integer Case

The case $\mathbb{I} = \mathbb{Z}$ is more complex!
The strong closure is not sufficient to provide the saturation. . .

**Theoretical Solution:**

Tight closure, proposed by [Harvey and Stuckey 97], can be used:

- $\forall i, j, k, \ m_{ij} \leq m_{ik} + m_{kj}$
- $\forall i, j, \ m_{ij} \leq (m_{i\bar{i}} + m_{\bar{j}j})/2$
- $\forall i, \ m_{i\bar{i}}$ is even (tightness)

Unfortunately, the normalisation algorithm runs in $O(n^4)$ . . .

**Open problem:** is there a $O(n^3)$ normalisation algorithm?

**Practical Solution:**

We use the strong closure and abandon the completeness, best, exactness results. In practice, we are precise enough.
Octagon Analysis Example: Absolute Value

\[ X \leftarrow [-100, 100] \]

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

The octagon domain can prove that, at 7, \[ -69 \leq X \leq 69. \]

1. \[ -100 \leq X \leq 100 \]
2. \[ -100 \leq X \leq 100 \ \land \ -100 \leq Y \leq 100 \ \land \ X - Y = 0 \ \land \ -200 \leq X + Y \leq 200 \]
3. \[ -100 \leq X \leq 0 \ \land \ -100 \leq Y \leq 0 \ \land \ X - Y = 0 \ \land \ -200 \leq X + Y \leq 0 \]
4. \[ -100 \leq X \leq 0 \ \land \ 0 \leq Y \leq 100 \ \land \ -200 \leq X - Y \leq 0 \ \land \ X + Y = 0 \]
5. \[ 0 \leq X \leq 100 \ \land \ 0 \leq Y \leq 100 \ \land \ X - Y = 0 \ \land \ 0 \leq X + Y \leq 200 \]
6. \[ -100 \leq X \leq 100 \ \land \ 0 \leq Y \leq 100 \ \land \ -200 \leq X - Y \leq 0 \ \land \ 0 \leq X + Y \leq 200 \]
7. \[ -69 \leq X \leq 69 \ \land \ 0 \leq Y \leq 69 \ \land \ -138 \leq X - Y \leq 0 \ \land \ 0 \leq X + Y \leq 138 \]

We require bounds on both \[ X - Y \] and \[ X + Y \]!
Octagon Analysis Example: Rate Limiter

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

\(Y\) is compelled to follow \(X\) and not change too rapidly: we have \(Y \in [-128, 128]\).

- The octagon domain can prove that \(|Y| \leq M\) is \textbf{stable} at \(\textcircled{1}\), if \(M \geq 144\).
  
  We need the \textbf{widening with thresholds} and \textbf{interval linear form assignment}.

- The polyhedron domain \textbf{can} prove that \(|Y| \leq 128\).

- The interval domain \textbf{cannot} prove any bound to be stable.
Static Variable Packing
Static Variable Packing

Problem:

Even a quadratic / cubic cost may be too costly in practice.

Solution:

Do not relate all the variables together!

- Split $\mathcal{V}$ into packs $\mathcal{V}_1, \ldots, \mathcal{V}_m \subseteq \mathcal{V}$.

- Associate one relational element per $\mathcal{V}_i$. We are relational inside packs and non-relational between packs.

- For flexibility, a variable may appear in several packs.

The packing defines a precision versus cost trade-off. The packing is static.
Static Variable Packing

Operator Adaptation

- Union, intersection, widening, etc. are defined **point-wisely**.

- For assignments, tests:
  - for each pack $\mathcal{V}_i$
    - **project** the expression on the set of variables $\mathcal{V}_i$
      - *(e.g., by replacing variables into intervals)*
    - apply the transfer function on $\mathcal{V}_i$

**Note:** We could perform inter-packing reduction using common variables as pivot... ... but we prefer to adapt the packing. (more predictable cost)

**Cost**

The cost depends only on:

- The size of each pack $|\mathcal{V}_1|, \ldots, |\mathcal{V}_m|$.
- The number of packs each variable appears in.

$\Longrightarrow$ It is interesting when there are **many small packs**.
Problem: How to determine a good packing $\mathcal{V}_1, \ldots, \mathcal{V}_m$?

Some ideas:

- Rely on variable scope. (does not help when many globals)
- Rely on variable occurring simultaneously in expressions.
- Rely on a previous analysis. (packing optimisation)

This must be done on a per programming style basis!

More of this when we will talk about Astrée...
Symbolic Enhancement Methods
Core Principle

**Idea:** Replace expressions with nicer ones on the fly.

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

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

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

The same holds for tests and backward assignments.

**Example Application:**

- Replace a non-linear assignment by a linear one. If $X \in [0, 1]$ in $\mathcal{X}^\#$, we replace $\{ V \leftarrow X \times Y \}(\mathcal{X}^\#)$ with $\{ V \leftarrow [0, 1] \times Y \}(\mathcal{X}^\#)$.

**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: \( [a_0, b_0] + \sum_k ([a_k, b_k] \times V_k) \).

**Interval Linear Form Manipulations:**

Resemble a vector space structure.

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

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

- \(\iota\left([[a_0, b_0] + \sum_k [a_k, b_k] \times V_k, x^\#]\right)\) def \(= [a_0, b_0] + \sum_k ([a_k, b_k] \times \pi_k(x^\#))\) (on-the-fly intervalisation)

We use interval arithmetics +, \times, and the interval projection \(\pi_k\).
Linearisation

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

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

In **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

Applications:

- Interval domain: linearisation provides **simplification for free**.

  Example: \((X + Y) - X\) where \(X, Y \in [0, 1]\).
  - without linearisation: \(\langle (X + Y) - X \rangle^\#(X^\#) = [-1, 2]\),
  - with linearisation: \(\langle (X + Y) - X \rangle(\langle X^\# \rangle)^\#(X^\#) = \langle Y \rangle^\#(X^\#) = [0, 1]\).

- Octagon domain: we can use our interval linear form transfer functions.

- We can abstract further into expressions of the form: \([a_0, b_0] + \sum_k c_k \times V_k\).
  This can be fed to the polyhedron domain.

The result greatly **depends on the chosen multiplication strategy**!

**Open problem:** find strategies with **theoretical precision guarantees**.
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: \( X^\# \in D^\# \) is enriched with a map \( S^\# \in (V \rightarrow E) \).

- Abstract elements \( \langle X^\#, S^\# \rangle \) now represent:
  \[ \gamma \langle X^\#, S^\# \rangle \overset{\text{def}}{=} \{ \rho \in \gamma(X^\#) \mid \forall i, \; \rho(V_i) \in \llbracket S^\#(V_i) \rrbracket(\rho) \}. \]

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

Note: We must choose how far to propagate.
Adaptation to Floating-Point
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**

![Sign Exponent e Fraction b](image)

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

- **normalised** numbers: $(-1)^s \times 2^{e-127} \times 1.b_1 \cdots b_{23}$ \hspace{1cm} ($1 \leq e \leq 254$)
- **denormalised** numbers: $(-1)^s \times 2^{e-126} \times 0.b_1 \cdots b_{23}$ \hspace{1cm} ($e = 0, b \neq 0$)
- **signed zeros**: $+0$ and $-0$
- **infinities and error codes**: $+\infty$, $-\infty$, $NaN$
IEEE 754-1985 Arithmetics

Floating-Point Expressions $\mathcal{E}_f$:

$$\mathcal{E}_f ::= [a, b] \quad \text{interval } a, b \in \mathbb{F}$$

$$\quad \quad X \quad \text{variable } X \in \mathcal{V}$$

$$\quad \quad \ominus \mathcal{E}_f \quad \text{unary operator}$$

$$\quad \quad \mathcal{E}_f \odot \mathcal{E}_f \quad \text{binary operators } \odot \in \{\oplus, \otimes, \ldots\}$$

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 $NaN$. 

NSAD’05 - Weakly Relational Numerical Abstract Domains - Antoine Miné 40/53
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** $\Omega$.
  $\implies$ 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.
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!

\[
(X \leq c) \land (Y \leq Z) \implies (X + Y \leq c + d)
\]

\[
(X \ominus Y \leq c) \land (Y \ominus Z \leq d) \nleftrightarrow (X \ominus Z \leq c \oplus d)
\]

\[
(10^{22} \oplus 1.000000019 \cdot 10^{38}) \ominus (10^{22} \ominus 1.000000019 \cdot 10^{38}) = 0
\]

**Solution:**

- \([e]_f\) is abstracted as a **linear interval form on** \(\mathbb{Q}\).

- Invariant semantics will be expressed **using** \(\mathbb{Q}\), \(+\), \(−\), \(\ldots\) not \(\mathbb{F}'\), \(\ominus\), \(\ominus\).

\(\implies\) 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 $\langle e \rangle_f$:

- $\langle e_1 \oplus e_2 \rangle_f(X^\#) \overset{\text{def}}{=} \langle e_1 \rangle_f(X^\#) \oplus \langle e_2 \rangle_f(X^\#) \cdot \varepsilon(\langle e_1 \rangle_f(X^\#)) \cdot \varepsilon(\langle e_2 \rangle_f(X^\#)) \cdot \omega$

- $\langle [a, b] \otimes e_2 \rangle_f(X^\#) \overset{\text{def}}{=} ([a, b] \otimes \langle e_2 \rangle_f(X^\#)) \cdot ([a, b] \otimes \varepsilon(\langle e_2 \rangle_f(X^\#))) \cdot \omega$

- 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
  \[
  \forall \rho \in \gamma(X^\#), \ [e]_f(\rho) \subseteq \llbracket (e)_{f(X^\#)} \rrbracket(\rho)
  \]
  and we can feed \( \{ V \leftarrow (e)_{f(X^\#)} \}^\# \) to an abstract domain in \( \mathbb{Q} \).

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

**Example:**

\[
\begin{align*}
Z \leftarrow X \ominus (0.25 \otimes X) & \quad \text{is linearised as} \\
Z \leftarrow ([0.749 \cdots, 0.750 \cdots] \times X) + (2.35 \cdots 10^{-38} \times [-1, 1])
\end{align*}
\]

- 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 Octagons

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,
- **octagon 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 \( \mathbb{F} \)**: 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., \( X \leftarrow 0.5X + [-1, 1] \),
- when concrete computations have explicit **limiters**,
- specific **widenings** and **narrowings** can help.

Some more theoretical work is needed to characterise the stability.
Application to 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 $3 \times 10^6$ times.
Numerical Abstract Domain Choice

Astrée uses the **octagon** domain preferably to the polyhedron domain because:

- it has a much **smaller asymptotic cost**, but also,
- it has **interval linear form operators** able to abstract float expressions,
- it can be easily **implemented using float numbers**.

Whenever possible, Astrée uses the **interval** domain with **linearisation** and **symbolic constant propagation** because it has a quasi-linear cost.

**Packing** is used to limit the use of non-linear cost domains.

All **relational** domains are built on top of our **floating-point linearisation**:

- the octagon domain,
- filters [Feret04] and arithmetic-geometric progression domains [Feret05].
Packing Results

There are too many variables even for the octagon domain \(\implies\) we use packing.

**Automatic Packing:** Using simple syntactic criteria

- associate one pack per syntactic block,
- put only variables related in the block’s expressions, ignoring sub-blocks,
- ignore obviously non-linear terms,
- relate variables in tests to both the directly enclosing and nested blocks.

**Results:**

<table>
<thead>
<tr>
<th># lines</th>
<th># variables</th>
<th># packs</th>
<th>avg. size</th>
<th>$\sqrt{\sum \text{size}^2}$</th>
<th>$\sqrt[3]{\sum \text{size}^3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>370</td>
<td>100</td>
<td>20</td>
<td>3.6</td>
<td>4.8</td>
<td>6.2</td>
</tr>
<tr>
<td>9 500</td>
<td>1 400</td>
<td>200</td>
<td>3.1</td>
<td>4.6</td>
<td>6.6</td>
</tr>
<tr>
<td>70 000</td>
<td>14 000</td>
<td>2 470</td>
<td>3.5</td>
<td>5.2</td>
<td>7.8</td>
</tr>
<tr>
<td>226 000</td>
<td>47 500</td>
<td>7 429</td>
<td>3.5</td>
<td>4.5</td>
<td>5.8</td>
</tr>
<tr>
<td>400 000</td>
<td>82 000</td>
<td>12 964</td>
<td>3.3</td>
<td>4.1</td>
<td>5.3</td>
</tr>
</tbody>
</table>

\(\implies\) Cost is a **linear** function of code size: the method is **scalable**.
## Analysis Results

On a 64-bit 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!

**Note:** Results date back from a few months; they have improved since.
Analysis Screenshot
Conclusion
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.
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!