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 $\Longrightarrow$ \$ 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.
- New numerical abstract domains: weakly relational domains.
- 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**

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

♦ **expressions**

$\mathcal{E}$	$::=$	$[a, b]$	interval $a \in \mathbb{I} \cup \{-\infty\}$, $b \in \mathbb{I} \cup \{+\infty\}$,
	$\mid$	X	variable $X \in \mathcal{V}$
	$\mid$	$- \mathcal{E}$	unary operator
	$\mid$	$\mathcal{E} \diamond \mathcal{E}$	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: $[\![\, \mathcal{E} \,]\!]: (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$

$\mathcal{E}$ maps **environments** to **sets** of numerical values:

$$
\begin{array}{lcl}
[\![\, [a,b] \,]\!](\rho) & \stackrel{\text{def}}{=} & \{\, c \in \mathbb{I} \mid a \le c \le b \,\}, \\
[\![\, X \,]\!](\rho) & \stackrel{\text{def}}{=} & \{\, \rho(X) \,\}, \\
[\![\, e_1 + e_2 \,]\!](\rho) & \stackrel{\text{def}}{=} & \{\, v_1 + v_2 \mid v_1 \in [\![\, e_1 \,]\!](\rho),\ v_2 \in [\![\, e_2 \,]\!](\rho) \,\}, \\
\text{etc.} & &
\end{array}
$$

Instruction Semantics: $\{\![\, \mathcal{I} \,]\!\}: \mathcal{P}(\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$

A **transfer function** defines a **relation** between environments:

$$
\begin{array}{lcl}
\{\![\, X \leftarrow e \,]\!\}(R) & \stackrel{\text{def}}{=} & \{\, \rho[\, X \mapsto v \,] \mid \rho \in R,\ v \in [\![\, e \,]\!](\rho) \,\}, \\
\{\![\, e \bowtie 0\ ? \,]\!\}(R) & \stackrel{\text{def}}{=} & \{\, \rho \in R \mid \exists v \in [\![\, e \,]\!](\rho) \text{ such that } v \bowtie 0 \,\}.
\end{array}
$$

Reachability Semantics

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

L	program points
$e \in L$	entry point
$I \subseteq L \times \mathcal{I} \times L$	arcs

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

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

that gathers all possible environments at each program points.

Problem: This is **not computable** in general.
$\Longrightarrow$ 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 $\mathcal{D}^\sharp$ of abstract values, together with:
- a concretisation: γ: $\mathcal{D}^\sharp \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$,
- a partial order: $\sqsubseteq^\sharp$, $\bot^\sharp$, $\top^\sharp$,
- sound, effective abstract transfer functions $\{\!| \mathcal{I} |\!\}^\sharp$: $(\{\!| \mathcal{I} |\!\} \circ \gamma)(\mathcal{X}^\sharp) \subseteq (\gamma \circ \{\!| \mathcal{I} |\!\}^\sharp)(\mathcal{X}^\sharp)$,
  a sound, effective abstract union $\cup^\sharp$: $\gamma(\mathcal{X}^\sharp) \cup \gamma(\mathcal{Y}^\sharp) \subseteq \gamma(\mathcal{X}^\sharp \cup^\sharp \mathcal{Y}^\sharp)$,
- effective extrapolation operators $\triangledown$, $\vartriangle$ if $\mathcal{D}^\sharp$ has infinite chains.

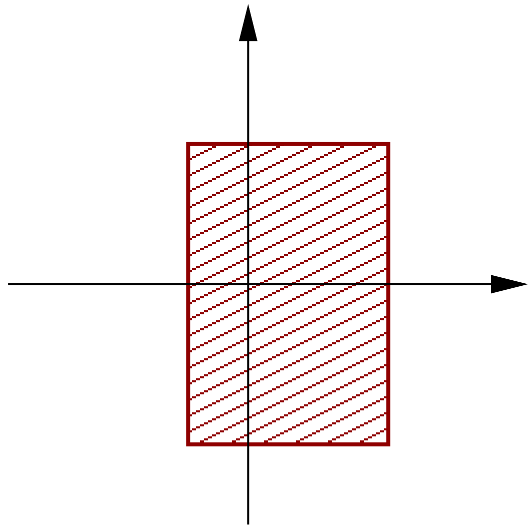
$\Longrightarrow$ we can perform a reachability analysis in $L \to \mathcal{D}^\sharp$ soundly.

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:

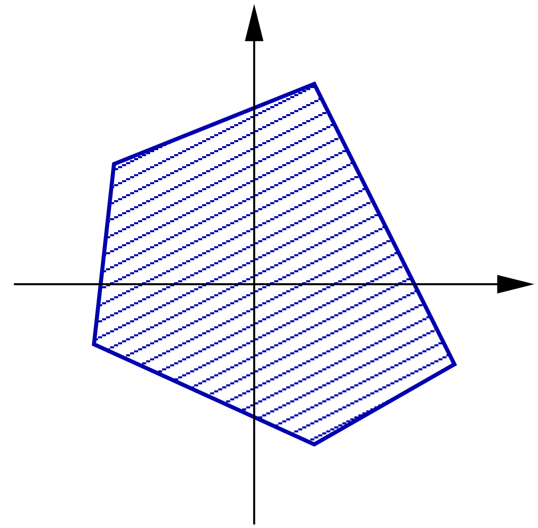


Intervals (1976)

$\bigwedge_i \ (X_i \in [a_i, b_i])$

non relational

linear cost



Polyhedra (1978)

$\bigwedge_j \ (\sum_i \alpha_{ij} X_i \leq \beta_j)$

relational

unbounded cost

exponential in practice

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 $\mathtt{V} \leq 11$ at •,
we need to prove the **relational** loop invariant $\mathtt{V} + \mathtt{I} \leq 10$ 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:

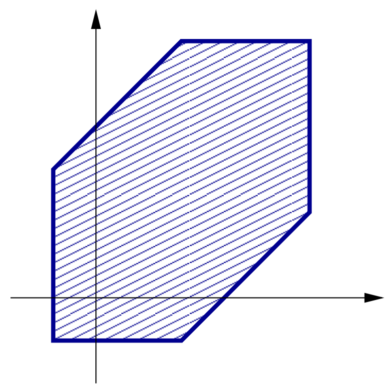
- zone abstract domain,
- octagon abstract domain,
- zone congruence abstract domain.

The Zone Abstract Domain

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

Zones enrich intervals with invariants of the form:

$$\bigwedge_{ij} (V_i - V_j \leq c_{ij}) \qquad c_{ij} \in \mathbb{I}$$



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.

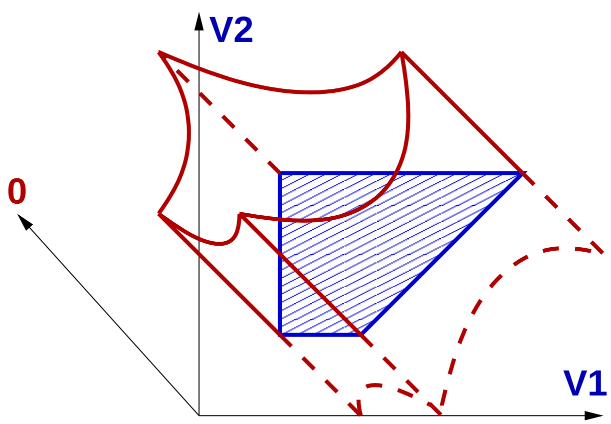
Zones are used in the model-checking of timed automata and Petri nets
but they need many new abstract operators to suit Abstract Interpretation needs.

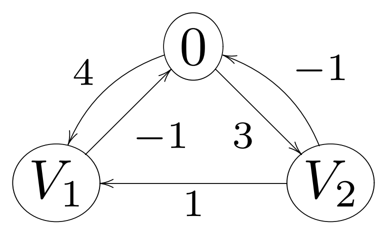
Zone Representation

Difference Bound Matrices: (DBMs)

- matrix of size $(n+1)\times(n+1)$ with elements in $\bar{\mathbb{I}} \stackrel{\text{def}}{=} \mathbb{I}\cup\{+\infty\}$:
  - $\mathbf{m}_{ij} \neq +\infty$ is an upper bound for $V_j - V_i$,
  - $\mathbf{m}_{ij} = +\infty$ means that $V_j - V_i$ is unbounded,
  - $\mathbf{m}_{i0}$, $\mathbf{m}_{0j}$ encode unary constraints: $-V_i \leq \mathbf{m}_{i0}$, $V_j \leq \mathbf{m}_{0j}$,
- $\gamma(\mathbf{m}) \stackrel{\text{def}}{=} \{\, (v_1,\ldots,v_n)\in\mathbb{I} \mid \forall i,j,\ v_j - v_i \leq \mathbf{m}_{ij},\ v_0 = 0 \,\}$,
- $\mathbf{m}$ is the adjacency matrix of a **weighted directed graph**: $V_i \xrightarrow{\mathbf{m}_{ij}} V_j$.

Example:





$$\begin{array}{cc|ccc} & & & j & \\ & & 0 & V_1 & V_2 \\ \hline & 0 & +\infty & 4 & 3 \\ i & V_1 & -1 & +\infty & +\infty \\ & V_2 & -1 & 1 & +\infty \end{array}$$

Order Structure

The total order on $\mathbb{I}$ is extended to $\overline{\mathbb{I}} \stackrel{\text{def}}{=} \mathbb{I} \cup \{+\infty\}$.

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

$$\begin{array}{cccl}
\mathbf{m} \sqsubseteq^\sharp \mathbf{n} & \stackrel{\text{def}}{\iff} & \forall i,j,\ \mathbf{m}_{ij} \leq \mathbf{n}_{ij} & \text{point-wise partial order} \\
\left[\mathbf{m} \sqcap^\sharp \mathbf{n}\right]_{ij} & \stackrel{\text{def}}{=} & \min(\mathbf{m}_{ij}, \mathbf{n}_{ij}) & \text{greatest lower bound} \\
\left[\mathbf{m} \sqcup^\sharp \mathbf{n}\right]_{ij} & \stackrel{\text{def}}{=} & \max(\mathbf{m}_{ij}, \mathbf{n}_{ij}) & \text{least upper bound} \\
\left[\top^\sharp\right]_{ij} & \stackrel{\text{def}}{=} & +\infty & \text{greatest element}
\end{array}$$

However:

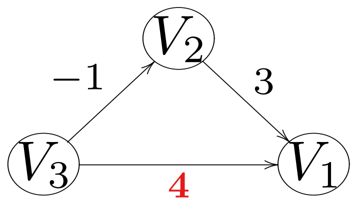
- $\mathbf{m} \sqsubseteq^\sharp \mathbf{n} \Longrightarrow \gamma(\mathbf{m}) \subseteq \gamma(\mathbf{n})$ but not the converse,
- $\mathbf{m} = \mathbf{n} \Longrightarrow \gamma(\mathbf{m}) = \gamma(\mathbf{n})$ but not the converse: γ is not injective!

$\Longrightarrow$ we introduce a **normal form**.

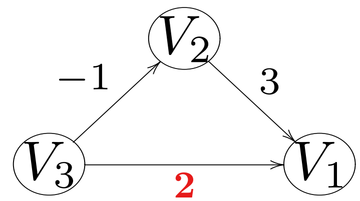
Normal Form

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

e.g. $\begin{cases} V_1 - V_2 \leq 3 \\ V_2 - V_3 \leq -1 \\ V_1 - V_3 \leq 4 \end{cases}$



$\Longrightarrow$



$\begin{cases} V_1 - V_2 \leq 3 \\ V_2 - V_3 \leq -1 \\ V_1 - V_3 \leq 2 \end{cases}$

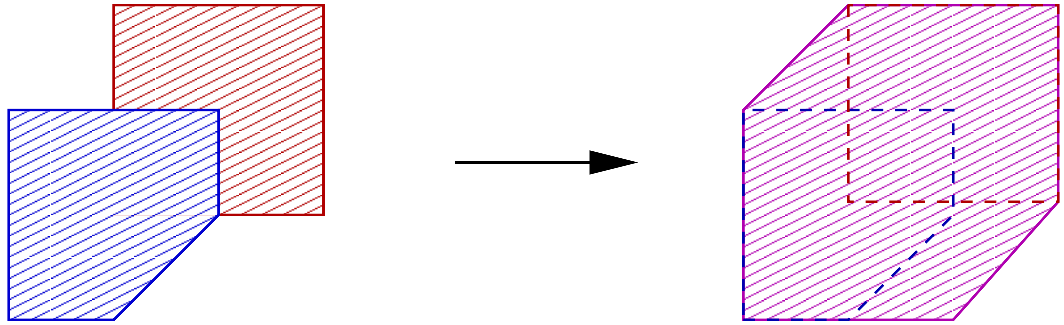
Shortest-Path Closure $\mathbf{m}^*$: Floyd–Warshall algorithm:

$$\begin{cases} \mathbf{m}^*_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}^{n+1}_{ij} \\ \mathbf{m}^0_{ij} & \stackrel{\text{def}}{=} & \mathbf{m}_{ij} \\ \mathbf{m}^{k+1}_{ij} & \stackrel{\text{def}}{=} & \min(\mathbf{m}^k_{ij},\ \mathbf{m}^k_{ik} + \mathbf{m}^k_{kj}) \quad \text{if } 0 \leq k \leq n \end{cases}$$

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

Operator Example: Abstract Union

The union of two zones is not always a zone:



$\sqcup^\sharp$ is a sound counterpart for $\cup$: $\gamma(\mathbf{m}) \cup \gamma(\mathbf{n}) \subseteq \gamma(\mathbf{m} \sqcup^\sharp \mathbf{n})$.
But it may not output the smallest zone encompassing two zones...
because of implicit constraints.

Solution: Define $\mathbf{m} \cup^\sharp \mathbf{n} \stackrel{\text{def}}{=} \mathbf{m}^* \sqcup^\sharp \mathbf{n}^*$:

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

Note: The intersection $\sqcap^\sharp$ 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 $X \leftarrow [a,b]$.

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

Interval and Polyhedra Based Assignments

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

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

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 polyhedron-based assignment is too costly, (exponential cost)
- the interval-based assignment is very imprecise. (not relational enough)

$\Longrightarrow$ 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:

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

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

$\Longrightarrow$ 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** $\triangledown$ to compute fixpoints in finite time:

$$\left\{\begin{array}{lcl}\mathcal{X}_0^\sharp & \stackrel{\text{def}}{=} & \mathcal{Y}_0^\sharp \\ \mathcal{X}_{i+1}^\sharp & \stackrel{\text{def}}{=} & \mathcal{X}_i^\sharp \triangledown \mathcal{Y}_{i+1}^\sharp\end{array}\right. \quad \text{should} \quad \left|\begin{array}{l}\text{converge in } \textbf{finite time} \\ \text{towards an over-approximation of } \bigcup_i \gamma(\mathcal{Y}_i^\sharp)\end{array}\right.$$

Example Widening:

Point-wise standard interval widening:

$$(\mathbf{m} \triangledown \mathbf{n})_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ +\infty & \text{otherwise} \end{cases}$$

Unstable constraints are simply thrown away.

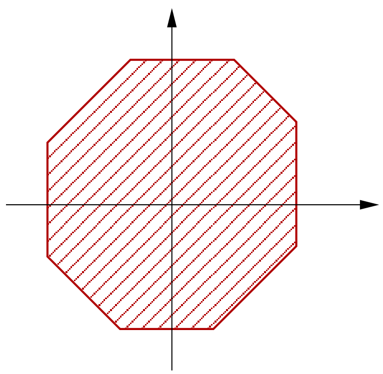
Notes:

- Any interval widening can be extended point-wisely.
- $\mathcal{X}_{i+1}^\sharp \stackrel{\text{def}}{=} (\mathcal{X}_i^{\sharp *}) \triangledown \mathcal{Y}_i^\sharp$ may diverge! Bad interaction between $*$ and $\triangledown$.

The Octagon Abstract Domain

Octagons extend zones to invariants of the form:

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



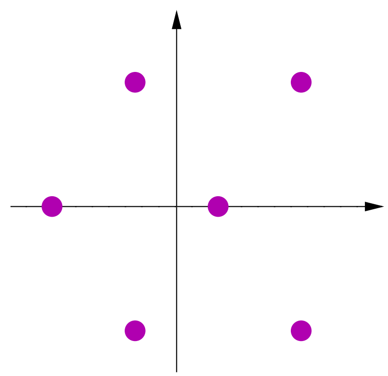
- It is strictly **more expressive** than the zone domain.
- It has the same asymptotic cost: **quadratic** in memory and **cubic** in time.
- It is sufficient to analyse our first example!

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:

$$\bigwedge_{ij} \left(V_i \equiv V_j + b_{ij} \; [c_{ij}] \right) \qquad \mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}\}$$



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 $\forall \rho \in \gamma(\mathcal{X}^\sharp)$, $[\![e]\!](\rho) \subseteq [\![e']\!](\rho)$, then:

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

$\Longrightarrow$ we can safely use $\{\!| V \leftarrow e' |\!\}^\sharp(\mathcal{X}^\sharp)$ in place of $\{\!| V \leftarrow e |\!\}^\sharp(\mathcal{X}^\sharp)$.

The same holds for tests.

Example Application:

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

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

Note: Interactions betwen numerical abstract values $\mathcal{X}^\sharp$ 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: ‘ , a , b , m , ι

Resemble a vector space structure.

- $([a_0, b_0] + \sum_k [a_k, b_k] \times V_k) \text{ ‘ } ([a'_0, b'_0] + \sum_k [a'_k, b'_k] \times V_k)$
  $\stackrel{\text{def}}{=} (([a_0, b_0] + [a'_0, b'_0]) + \sum_k ([a_k, b_k] + [a'_k, b'_k]) \times V_k)$

- $[a, b] \text{ b } ([a'_0, b'_0] + \sum_k [a'_k, b'_k] \times V_k) \stackrel{\text{def}}{=} ([a, b] \times [a'_0, b'_0]) + \sum_k ([a, b] \times [a'_k, b'_k]) \times V_k$

- $\iota([a_0, b_0] + \sum_k [a_k, b_k] \times V_k, \mathcal{X}^\sharp) \stackrel{\text{def}}{=} [a_0, b_0] + \sum_k ([a_k, b_k] \times \pi_k(\mathcal{X}^\sharp))$

(on-the-fly intervalisation)

Linearisation (continued)

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

- $(\!| \, V_i \, |\!)(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} [1,1] \times V_i$
- $(\!| \, e_1 + e_2 \, |\!)(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} (\!| \, e_1 \, |\!)(\mathcal{X}^\sharp) \; \text{‘} \; (\!| \, e_2 \, |\!)(\mathcal{X}^\sharp)$
- $(\!| \, e_1 \times e_2 \, |\!)(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} [a,b] \; \mathrm{b} \; (\!| \, e_2 \, |\!)(\mathcal{X}^\sharp)$ when $(\!| \, e_1 \, |\!)(\mathcal{X}^\sharp) = [a,b]$
- $(\!| \, e_1 \times e_2 \, |\!)(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} \iota((\!| \, e_1 \, |\!)(\mathcal{X}^\sharp), \mathcal{X}^\sharp) \; \mathrm{b} \; (\!| \, e_2 \, |\!)(\mathcal{X}^\sharp)$ or
  $(\!| \, e_1 \times e_2 \, |\!)(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} \iota((\!| \, e_2 \, |\!)(\mathcal{X}^\sharp), \mathcal{X}^\sharp) \; \mathrm{b} \; (\!| \, e_1 \, |\!)(\mathcal{X}^\sharp)$

Notes:

- Non-linear multiplication: we must **choose** whether to intervalise e_1 or e_2.
  example: intervalise the expression with smallest bounds
  $$X \in [0,1],\; Y \in [-10,10] \Longrightarrow (\!| \, X \times Y \, |\!)(\mathcal{X}^\sharp) = [0,1] \times Y$$

- Linearisation provides **simplification for free**: $(\!| \, (X+Y) - X \, |\!)(\mathcal{X}^\sharp) = Y$.

  If $X, Y \in [0,1]$, interval arithmetics gives $[\![\, (X+Y) - X \,]\!]^\sharp = [-1,2]$ but $[\![\, Y \,]\!]^\sharp = [0,1]$.

Symbolic Constant Propagation

Idea: Enhance simplification-by-linearisation using expression propagation.

Example: $X \leftarrow Y + Z; U \leftarrow X - Z$

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

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

- Abstract elements $< \mathcal{X}^\sharp, S^\sharp >$ now represent:

$$\gamma < \mathcal{X}^\sharp, S^\sharp > \stackrel{\text{def}}{=} \{\ \rho \in \gamma(\mathcal{X}^\sharp) \mid \forall i,\ \rho(V_i) \in [\![\ S^\sharp(V_i)\ ]\!](\rho)\ \}.$$

- Abstract assignments $\{\!| X \leftarrow e |\!\}^\sharp < \mathcal{X}^\sharp, S^\sharp >$
  - **propagate** $S^\sharp$ into e to get e' and evaluate $\mathcal{X}^{\sharp\prime} \stackrel{\text{def}}{=} \{\!| X \leftarrow e' |\!\}^\sharp(\mathcal{X}^\sharp)$,
  - kill information on X in $S^\sharp$, 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 $\mathbb{F}$

Sign	Exponent e			Fraction b		
s	e_8	$\cdots$	e_1	b_1	$\cdots$	b_{23}

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

- **normalised** numbers: $(-1)^s \times 2^{e-127} \times \mathbf{1}.b_1 \cdots b_{23}$ $\qquad (1 \leq e \leq 254)$
- **denormalised** numbers: $(-1)^s \times 2^{-126} \times \mathbf{0}.b_1 \cdots b_{23}$ $\qquad (e = 0,\ b \neq 0)$
- signed **zeros**: $+0$ and -0 $\qquad$ (if $e = 0,\ b = 0$)
- infinities and error codes: $+\infty$, $-\infty$, *NaN* $\qquad$ (if $e = 255$)

IEEE 754-1985 Arithmetics

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

$$
\begin{array}{rcll}
\mathcal{E}_\mathrm{f} & ::= & [a, b] & \text{interval } a, b \in \mathbb{F} \\
& | & X & \text{variable } X \in \mathcal{V} \\
& | & \ominus\, \mathcal{E}_f & \text{unary operator} \\
& | & \mathcal{E}_f \odot \mathcal{E}_f & \text{binary operators } \odot \in \{\oplus, \otimes, \ldots\}
\end{array}
$$

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*.

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

Partial Definition of $[\![\, e \,]\!]_f$: (with rounding towards $+\infty$)

- $[\![\, e_1 \oplus e_2 \,]\!]_f(\rho) \stackrel{\text{def}}{=} \{\, R(v_1 + v_2) \mid v_1 \in [\![\, e_1 \,]\!]_f(\rho),\ v_2 \in [\![\, e_2 \,]\!]_f(\rho) \,\}$,
- etc.
- $R(x) \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } x = \Omega \text{ or } x > 2^{127}(2 - 2^{-23}) \\ \min \{y \in \mathbb{F}' \mid y \geq x\} & \text{otherwise} \end{cases}$
- 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!

  e.g. $$(X - Y \leq c) \wedge (Y - Z \leq d) \Longrightarrow (X - Z \leq c + d)$$ (Zone propagation)

  $$(X \ominus Y \leq c) \wedge (Y \ominus Z \leq d) \not\Longrightarrow (X \ominus Z \leq c \oplus d)$$
  $$(10^{22} \oplus 1.000000019 \cdot 10^{38}) \;\oplus\; (10^{22} \ominus 1.000000019 \cdot 10^{38}) = 0$$

Solution:

- $[\![\, e \,]\!]_f$ is abstracted as a **linear interval form** on $\mathbb{Q}$.
- Invariant semantics will be expressed **using** $\mathbb{Q}$, $+$, $-$, $\ldots$ not $\mathbb{F}'$, $\oplus$, $\ominus$.
  $\Longrightarrow$ 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** ε of amplitude 2^{-23}, expressed as a linear form:

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

  (normalised numbers)

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

$\Longrightarrow$ We sum these two causes of rounding.

Linearisation $(\!|\, \mathbf{e} \,|\!)_{\mathbf{f}}$:

- $(\!|\, e_1 \oplus e_2 \,|\!)_f(\mathcal{X}^\sharp) \stackrel{\text{def}}{=}$

  $(\!|\, e_1 \,|\!)_f(\mathcal{X}^\sharp)$ ‘ $(\!|\, e_2 \,|\!)_f(\mathcal{X}^\sharp)$ ‘ $\varepsilon((\!|\, \mathbf{e_1} \,|\!)_{\mathbf{f}}(\mathcal{X}^\sharp))$ ‘ $\varepsilon((\!|\, \mathbf{e_2} \,|\!)_{\mathbf{f}}(\mathcal{X}^\sharp))$ ‘ ω

- $(\!|\, [a,b] \otimes e_2 \,|\!)_f(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} ([a,b]$ b $(\!|\, e_2 \,|\!)_f(\mathcal{X}^\sharp))$ ‘ $([a,b]$ b $\varepsilon((\!|\, \mathbf{e_2} \,|\!)_{\mathbf{f}}(\mathcal{X}^\sharp)))$ ‘ ω

- etc.

Application of Floating-Point Linearisation

Abstract Assignment: $V \leftarrow e$

We first evaluate $\mathbf{e}$ in the floating-point interval domain.

- If there is no run-time error Ω detected, then
$$\forall \rho \in \gamma(\mathcal{X}^\sharp),\ [\![\, e \,]\!]_f(\rho) \subseteq [\![\, (\!| e |\!)_f(\mathcal{X}^\sharp) \,]\!](\rho)$$
and we can **feed** $\{\!| V \leftarrow (\!| e |\!)_f(\mathcal{X}^\sharp) |\!\}^\sharp$ to an abstract domain in $\mathbb{Q}$.

- If Ω 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 $\mathbb{Q}$.
$\Longrightarrow$ 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 $\mathbb{F}$:** 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.

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.

# lines	# variables	# packs	pack size
370	100	20	3.6
9 500	1 400	200	3.1
70 000	14 000	2 470	3.5
226 000	47 500	7 429	3.5
400 000	82 000	12 964	3.3

$\Longrightarrow$ **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

	without symbolic			**without octagon**			**with everything**		
# lines	time	memory	alarms	time	memory	alarms	time	memory	alarms
370	1.8s	16 MB	0	1.7s	14 MB	0	3.1s	16 MB	0
9 500	90s	81 MB	8	75s	75 MB	8	160s	80 MB	8
70 000	2h 40mn	559 MB	391	3h 17mn	537 MB	58	1h 16mn	582 MB	0
226 000	11h 16mn	1.3 GB	141	7h 8mn	1.0 GB	165	8h 5mn	1.3 GB	1
400 000	22h 8mn	2.2 GB	282	20h 31mn	1.7 GB	804	**13h 52mn**	**2.2 GB**	**0**

$\Longrightarrow$ **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

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

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!

Appendices

Octagon Analysis Example (1)

Absolute Value Computation:

$$
\begin{array}{ll}
 & X \leftarrow [-100, 100] \\
① & Y \leftarrow X \\
② & \texttt{if } Y \leq 0 \texttt{ \{ } ③ \ \ Y \leftarrow -Y \ ④ \texttt{ \} else \{ } ⑤ \texttt{ \}} \\
⑥ & \texttt{if } Y \leq 69 \texttt{ \{ } ⑦ \ \cdots \texttt{ \}}
\end{array}
$$

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

$$
\begin{array}{ll}
① & -100 \leq X \leq 100 \\
② & -100 \leq X \leq 100 \ \wedge \ -100 \leq Y \leq 100 \ \wedge \ X - Y = 0 \ \wedge \ -200 \leq X + Y \leq 200 \\
③ & -100 \leq X \leq 0 \ \wedge \ -100 \leq Y \leq 0 \ \wedge \ X - Y = 0 \ \wedge \ -200 \leq X + Y \leq 0 \\
④ & -100 \leq X \leq 0 \ \wedge \ 0 \leq Y \leq 100 \ \wedge \ -200 \leq X - Y \leq 0 \ \wedge \ X + Y = 0 \\
⑤ & 0 \leq X \leq 100 \ \wedge \ 0 \leq Y \leq 100 \ \wedge \ X - Y = 0 \ \wedge \ 0 \leq X + Y \leq 200 \\
⑥ & -100 \leq X \leq 100 \ \wedge \ 0 \leq Y \leq 100 \ \wedge \ -200 \leq X - Y \leq 0 \ \wedge \ 0 \leq X + Y \leq 200 \\
⑦ & -69 \leq X \leq 69 \ \wedge \ \mathbf{0 \leq Y \leq 69} \ \wedge \ \mathbf{-138 \leq X - Y \leq 0} \ \wedge \ \mathbf{0 \leq X + Y \leq 138}
\end{array}
$$

Octagon Analysis Example (2)

Rate Limiter:

```
    Y ← 0
    while ① random() {
      X ← [−128, 128]
      D ← [0, 16]
      S ← Y
②     R ← X − S
      Y ← X
      if R ≤ −D { ③ Y ← S − D ④ } else
      if D ≤ R  { ⑤ Y ← S + D ⑥ }
⑦   }
```

The octagon domain can prove that $|Y| \leq M$ is **stable** at ① for any $M \geq 144$.

In fact, we have $Y \in [-128, 128]\ldots$

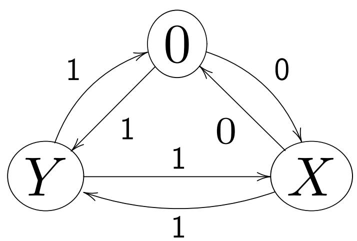
Note: The interval domain **cannot** prove any bound to be stable.

Interaction Between Closure and Widening

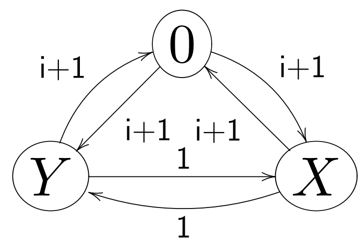
```
X ← 0
Y ← [−1, 1]
while random() {
  if X = Y {
    if random() { Y ← X + [−1, 1] }
    else        { X ← Y + [−1, 1] }
  }
}
```

Non-Terminating Analysis: Using $\mathbf{m}_{i+1} \stackrel{\text{def}}{=} (\mathbf{m}_i{}^*) \mathbin{\triangledown} \mathbf{n}_i$

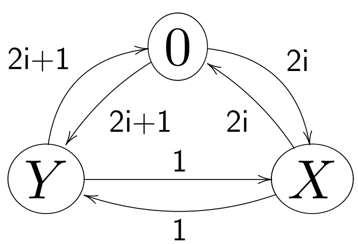
$\mathbf{m}_0 \stackrel{\text{def}}{=}$



$\mathbf{n}_i \stackrel{\text{def}}{=}$



$\mathbf{m}_{2i} =$



$\mathbf{m}_{2i+1} =$