Using CSP refutation capabilities to refine AI-based Approximations

Michel RUEHER

University of Nice Sophia-Antipolis / I3S – CNRS, France

(joined work with Olivier Ponsini, Claude Michel)

September, 2011

Dagstuhl-Seminar 11371

“Uncertainty modeling and analysis with intervals”
Outline

Verifying programs with floating-point computations
   Context
   Problems with floating-point numbers
   Objective & Approach
   Example

Abstract Interpretation & Fluctuat
   Static analyzer
   Zonotopes

Refining approximations with CP
   Overview
   Details

Experiments
   Programs
   Results over the reals
   Results over the floating-point numbers

Conclusion
Verifying programs with floating-point computations

Context

- Embedded Systems (transportation, nuclear energy...) rely more and more on floating-point computations

- C language is widely used for such applications

- Floats → an additional source of errors
Problems with floating-point numbers

• **Counter intuitive Properties** and “pitfalls” of Floating-point arithmetic:
  • Arithmetic operators are neither associative nor distributive
  • Reasoning with rounding, absorption, cancellation

• **Examples** (in simple precision)
  • Absorption: $10^7 + 0.5 = 10^7$
  • Cancellation: $((1 - 10^{-7}) - 1) \times 10^7 = -1.192... (\neq 1)$
  • $(10000001 - 10^7) + 0.5 \neq 10000001 - (10^7 + 0.5)$
  • $0.1 = (0.000110011001100\ldots)$
Semantics of program with floating-point numbers

Programs are run on the floats but:

• **Specification, properties** of programs
  ↦ Users are *reasoning with real numbers*

• **Programs** are sometimes written with the semantics of real numbers “in mind”

• **Differences** between computations over real numbers and computations over the floats
  → reveal **problems with floats**

Abstract Interpretation

→ **Approximations** of computations over *floats* and computations over the *real numbers*
Objective & Approach

• **Goal:** Refine the approximations of the domains of the program variables computed by abstract interpretation

• **Approach:** Use local consistencies to “shave” the domains
  → More accurate “semantics”
  → Complementary to “abstract domains”
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0, 10];
float y = x * x - x;
if (y >= 0)
  y = x / 10;
else
  y = x * x + 2;
y = 1 / (y - 1.5);
```

Diagram:

- $P_0: (D_x^0, D_y^0)$
- $y = x \times x - x$
- $y \geq 0$
- $y < 0$
- $P_1$
- $P_2$
- $P_4$
- $y = x / 10$
- $y = x \times x + 2$
- $P_3$
- $P_5$
- $P_6$
- $P_7$
- $y = 1 / (y - 1.5)$
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```csharp
float x = [0, 10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
```
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0,10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
```
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A **semantics** to compute the states of a program at various checkpoints
2. An **abstraction** to represent the states of a program
3. A **fixed point computation** of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0, 10];
float y = x * x - x;
if (y >= 0)
    y = x / 10;
else
    y = x * x + 2;
y = 1 / (y - 1.5);
```

```
\[ P_0: ([0, 10], [-\infty, +\infty]) \]
\[ y = x \times x - x \]
\[ P_1: \quad D_y^1 = [-10, 100] \]
\[ y \geq 0 \quad y < 0 \]
\[ P_2: \quad D_y^2 = D_y^1 \cap [0, +\infty] \]
\[ y = x / 10 \]
\[ P_3 \]
\[ P_4 \]
\[ P_5 \]
\[ y = x \times x + 2 \]
\[ P_6 \]
\[ P_7 \]
\[ y = 1 / (y - 1.5) \]
```
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A **semantics** to compute the states of a program at various checkpoints

2. An **abstraction** to represent the states of a program

3. A **fixed point computation** of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0, 10];
float y = x * x - x;
if (y >= 0)
    y = x/10;
else
    y = x * x + 2;
y = 1 / (y - 1.5);
```

<table>
<thead>
<tr>
<th><strong>P₀</strong></th>
<th><strong>P₁</strong></th>
<th><strong>P₂</strong></th>
<th><strong>P₃</strong></th>
<th><strong>P₄</strong></th>
<th><strong>P₅</strong></th>
<th><strong>P₆</strong></th>
<th><strong>P₇</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P₀</strong>: ([0, 10], [−∞, +∞])</td>
<td><strong>P₁</strong>: ( D_y^1 = [−10, 100] )</td>
<td><strong>P₂</strong>: ( D_y^2 = [0, 100] )</td>
<td><strong>P₃</strong></td>
<td><strong>P₄</strong></td>
<td><strong>P₅</strong></td>
<td><strong>P₆</strong></td>
<td><strong>P₇</strong></td>
</tr>
<tr>
<td>( y = x * x - x )</td>
<td>( y \geq 0 )</td>
<td>( y &lt; 0 )</td>
<td>( y = x/10 )</td>
<td>( y = x * x + 2 )</td>
<td>( y = 1 / (y - 1.5) )</td>
<td>( y = 1 / (y - 1.5) )</td>
<td></td>
</tr>
</tbody>
</table>
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0, 10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y - 1.5);
```

\[ P_0 : ([0, 10], [-\infty, +\infty]) \]

\[ y = x \times x - x \]

\[ P_1 : D^1_y = [-10, 100] \]

\[ y \geq 0 \]

\[ y < 0 \]

\[ P_2 : D^2_y = [0, 100] \]

\[ y = x/10 \]

\[ P_4 \]

\[ y = x \times x + 2 \]

\[ P_3 : D^3_y = D^0_x / [10, 10] \]

\[ y = 1 / (y - 1.5) \]

\[ P_5 \]

\[ P_6 \]

\[ P_7 \]
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A **semantics** to compute the states of a program at various checkpoints
2. An **abstraction** to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0, 10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
```

```
P₀: ([0, 10], [−∞, +∞])
P₁: D₁ = [−10, 100]
    y ≥ 0
    y < 0
P₂: D₂ = [0, 100]
    y = x/10
    y = x * x + 2
P₃: D₃ = [0, 1]
    y = 1 / (y − 1.5)
P₄
P₅
P₆
P₇
```
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0,10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
```

```
\begin{align*}
P_0 & : ([0, 10], [-\infty, +\infty]) \\
\end{align*}
```

```
\begin{align*}
P_1 : D_y^1 &= [-10, 100] \\
P_2 : D_y^2 &= [0, 100] \\
P_3 : D_y^3 &= [0, 1] \\
P_4 : D_y^4 &= [-10, 0] \\
P_5 : D_y^5 &= [-10, 0] \\
P_6 & : D_y^6 = [0, 1] \\
P_7 & : D_y^7 = [0, 1]
\end{align*}
```

```
y = x*x - x
y \geq 0
y < 0
P_0: ([0, 10], [-\infty, +\infty])

P_1: D_y^1 = [-10, 100]

P_2: D_y^2 = [0, 100]

P_3: D_y^3 = [0, 1]

P_4: D_y^4 = [-10, 0]

P_5:

P_6:

P_7:

y = 1 / (y - 1.5)

y = x / 10

y = x * x + 2
```
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

float x = [0, 10];
float y = x*x - x;
if (y >= 0)
y = x/10;
else
y = x*x + 2;
y = 1 / (y-1.5);
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. A **semantics** to compute the states of a program at various checkpoints
2. An **abstraction** to represent the states of a program
3. A **fixed point computation** of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0,10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
```

```
$P_0: ([0, 10], [-\infty, +\infty])$

$y = x \times x - x$

$P_1: D_y^1 = [-10, 100]$

$y \geq 0$

$P_2: D_y^2 = [0, 100]$

$y < 0$

$P_3: D_y^3 = [0, 1]$

$P_4: D_y^4 = [-10, 0[$

$y = x \times x + 2$

$P_5: D_y^5 = [2, 102]$

$y = 1/(y - 1.5)$

$P_6$

$P_7$
```
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

```c
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```

**Equations**

- $P_0$: $([0, 10], [-\infty, +\infty])$
- $y = x \times x - x$
- $P_1$: $D_y^1 = [-10, 100]$
- $y \geq 0$
- $P_2$: $D_y^2 = [0, 100]$
- $y < 0$
- $P_3$: $D_y^3 = [0, 1]$
- $P_4$: $D_y^4 = [-10, 0[$
- $y = x \times x + 2$
- $P_5$: $D_y^5 = [2, 102]$
- $P_6$: $D_y^6 = D_y^3 \cup D_y^5$
- $y = 1/(y - 1.5)$
- $P_7$: $D_y^7 = D_y^6$
Example (Abstract Interpretation)

Abstract Interpretation requires:
1. A semantics to compute the states of a program at various checkpoints
2. An abstraction to represent the states of a program
3. A fixed point computation of the equations of the semantics

Example (abstract domain of intervals)

float x = [0,10];
float y = x*x - x;
if (y >= 0)
    y = x/10;
else
    y = x*x + 2;
y = 1 / (y-1.5);
**Example (Abstract Interpretation)**

Abstract Interpretation requires:

1. A **semantics** to compute the states of a program at various checkpoints
2. An **abstraction** to represent the states of a program
3. A **fixed point computation** of the equations of the semantics

**Example (abstract domain of intervals)**

```plaintext
float x = [0,10];
float y = x * x - x;
if (y >= 0)
    y = x / 10;
else
    y = x * x + 2;
y = 1 / (y - 1.5);
```

---

**States**

- $P_0: (\mathbb{D}_0x, \mathbb{D}_0y) = ([0, 10], [-\infty, +\infty])$
- $P_1: D_y^1 = [-10, 100]$
- $P_2: D_y^2 = [0, 100]$
- $P_3: D_y^3 = [0, 1]$
- $P_4: D_y^4 = [-10, 0]$
- $P_5: D_y^5 = [2, 102]$
- $P_6: D_y^6 = [0, 102]$
- $P_7: D_y^7 = \frac{[1,1]}{D_y^6-[1.5,1.5]}$

---

**Equation**

$y = x \times x - x$

**Invariants**

- $y \geq 0$
- $y < 0$
- $y = x / 10$
- $y = x \times x + 2$
- $y = 1 / (y - 1.5)$
Example (Abstract Interpretation)

Abstract Interpretation requires:

1. **A semantics** to compute the states of a program at various checkpoints
2. **An abstraction** to represent the states of a program
3. **A fixed point computation** of the equations of the semantics

Example (abstract domain of intervals)

```plaintext
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
```
Example (constraint-based approach)

```java
float x = [0,10];
float y = x * x - x;
if (y >= 0)
    y = x / 10;
else
    y = x * x + 2;
y = 1 / (y - 1.5);
```

**Constraints:**

- \( y_0 = x_0 \times x_0 - x_0 \)
- If \( y_0 \geq 0 \)
  - \( y_1 = x_0 / 10 \)
- Else
  - \( y_1 = x_0 \times x_0 + 2 \)
- \( y_2 = 1 / (y_1 - 1.5) \)

**Domains:**

- \( P_0: (D^0_{x_0}, D^0_{y_0}, D^0_{y_1}, D^0_{y_2}) \)
- \( P_1: (D^0_{x_0}, D^0_{y_0}) \)
- \( P_2: (D^0_{y_0}, D^0_{y_1}) \)
- \( P_3: (D^0_{y_1}, D^0_{y_2}) \)
- \( P_4: (D^0_{y_0}) \)
- \( P_5: (D^0_{y_1}) \)
- \( P_6: (D^0_{y_2}) \)
- \( P_6': (D^0_{y_2}) \)
- \( P_7: (D^0_{y_2}) \)
Example (constraint-based approach)

float $x_0 = [0, 10]$;
float $y_0 = x_0 \times x_0 - x_0$;
if ($y_0 \geq 0$)
  $y_1 = x_0 / 10$;
else
  $y_1 = x_0 \times x_0 + 2$;
$y_2 = 1 / (y_1 - 1.5)$;
Example (constraint-based approach)

```plaintext
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

```plaintext
P0: ([0, 10], [−∞, ∞], [−∞, ∞], [−∞, ∞])

P1: y0 ≥ 0
P2: y1 = x0 / 10
P3: y2 = 1 / (y1 - 1.5)

P4: y0 < 0
P5: y1 = x0 * x0 + 2
P6: P6': y2 = 1 / (y1 - 1.5)

P7: P7: P7: P7: P7:
```
Example (constraint-based approach)

```plaintext
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

```
\begin{align*}
y_0 &= x_0 \times x_0 - x_0 \\
y_0 &= x_0 \times x_0 - x_0 \\
y_0 \geq 0 &\quad y_0 < 0 \\
P_0: ([0, 10], [-\infty, \infty], \\
\quad [-\infty, \infty], [-\infty, \infty]) \\
P_1: ([0, 10], [-10, 100], \\
\quad D^{0}_{y_1}, D^{0}_{y_2}) \\
P_2 &
\quad y_1 = x_0 / 10 \\
P_3 &
\quad y_1 = x_0 \times x_0 + 2 \\
P_4 &
\quad y_2 = 1 / (y_1 - 1.5) \\
P_5 &
\quad y_2 = 1 / (y_1 - 1.5) \\
P_6 &
\quad y_2 = 1 / (y_1 - 1.5) \\
P_6' &
\quad y_2 = 1 / (y_1 - 1.5) \\
P_7 &
\quad y_2 = 1 / (y_1 - 1.5)
\end{align*}
```
Example (constraint-based approach)

```c
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

\[ y_0 = x_0 \times x_0 - x_0 \]
\[ y_0 \geq 0 \]
\[ y_0 < 0 \]
\[ y_1 = x_0 / 10 \]
\[ y_1 = x_0 \times x_0 + 2 \]
\[ y_2 = 1 / (y_1 - 1.5) \]
\[ y_2 = 1 / (y_1 - 1.5) \]
Example (constraint-based approach)

```plaintext
float x_0 = [0,10];
float y_0 = x_0 * x_0 - x_0;
if (y_0 >= 0)
    y_1 = x_0 / 10;
else
    y_1 = x_0 * x_0 + 2;
Y_2 = 1 / (y_1 - 1.5);
y_0 = x_0 * x_0 - x_0
```

**Problematic AI & Fluctuation**

Refining approximations with CP

Experiments

Conclusion

**Example (constraint-based approach)**

- **P0**: ([0, 10], [−∞, ∞], [−∞, ∞], [−∞, ∞])
- **P1**: ([0, 10], [−10, 100], D_{y_1}^0, D_{y_2}^0)
- **P2**: ([0, 10], [0, 100], D_{y_1}^0, D_{y_2}^0)
- **P3**: ([0, 10], [0, 100], [0, 1], D_{y_2}^0)
- **P4**: ([0, 1], D_{y_1}^0, D_{y_2}^0)
- **P5**: ([0, 1], D_{y_1}^0, D_{y_2}^0)
- **P6**: ([0, 1], D_{y_1}^0, D_{y_2}^0)
- **P6'**: ([0, 1], D_{y_1}^0, D_{y_2}^0)
- **P7**: ([0, 1], D_{y_1}^0, D_{y_2}^0)
Example (constraint-based approach)

\[ y_0 = x_0 \times x_0 - x_0 \]

\[ y_0 \geq 0 \quad y_0 < 0 \]

float \( x_0 = [0, 10] \);
float \( y_0 = x_0 \times x_0 - x_0 \);

if \((y_0 \geq 0)\)
\( y_1 = x_0 / 10 \);
else  
\( y_1 = x_0 \times x_0 + 2 \);
\( y_2 = 1 / (y_1 - 1.5) \);

\( P_0: ([0, 10], [-\infty, \infty], [-\infty, \infty], [-\infty, \infty]) \)
\( P_1: ([0, 10], [-10, 100], D_{y_1}^0, D_{y_2}^0) \)
\( P_2: ([0, 10], [0, 100], D_{y_1}^0, D_{y_2}^0) \)
\( P_3: ([0, 10], [0, 100], [0, 1], D_{y_2}^0) \)
\( P_4: ([0, 10], [0, 100], [0, 1], D_{y_2}^0) \)
\( P_5: ([0, 10], [0, 100], [0, 1], [-2, -0.667]) \)
Example (constraint-based approach)

float \( x_0 = [0, 10] \);
float \( y_0 = x_0 \cdot x_0 - x_0 \);
if \( y_0 \geq 0 \)
  \( y_1 = x_0 / 10 \);
else
  \( y_1 = x_0 \cdot x_0 + 2 \);
\( y_2 = 1 / (y_1 - 1.5) \);

\( P_0: ([0, 10], [-\infty, \infty], [-\infty, \infty], [-\infty, \infty]) \)
\( P_1: ([0, 10], [-10, 100], D_{y_1}^0, D_{y_2}^0) \)
\( P_2: ([0, 10], [0, 100], D_{y_1}^0, D_{y_2}^0) \)
\( P_3: ([0, 10], [0, 100], [0, 1], D_{y_2}^0) \)
\( P_4: ([0, 1], [-1, 0], D_{y_1}^0, D_{y_2}^0) \)
\( P_5 \)
\( P_6: ([0, 10], [0, 100], [0, 1], [-2, -0.667]) \)
\( P_6' \)
\( P_7 \)
\( y_0 = x_0 \cdot x_0 - x_0 \)
\( y_0 \geq 0 \)
\( y_0 < 0 \)
\( y_1 = x_0 / 10 \)
\( y_1 = x_0 \cdot x_0 + 2 \)
\( y_2 = 1 / (y_1 - 1.5) \)
\( y_2 = 1 / (y_1 - 1.5) \)
Example (constraint-based approach)

```plaintext
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

- **P0**: \( ([0, 10], [-\infty, \infty], [-\infty, \infty], [-\infty, \infty]) \)
- **P1**: \( ([0, 10], [-10, 100], \{D_y^0, D_y^0\}) \)
- **P2**: \( ([0, 10], [0, 100], \{D_y^0, D_y^0\}) \)
- **P3**: \( ([0, 10], [0, 100], \{0, 1\}, \{D_y^0\}) \)
- **P4**: \( ([0, 1], [-1, 0], \{D_y^0, D_y^0\}) \)
- **P5**: \( ([0, 1], [-1, 0], \{2, 3.001\}, \{D_y^0\}) \)
- **P6**: \( ([0, 10], [0, 100], \{0, 1\}, \{-2, -0.667\}) \)
- **P6'**: \( ([0, 10], [0, 100], \{0, 1\}, \{-2, -0.666\}) \)
- **P7**: \( ([0, 1], [-2, 2], \{D_y^2\}) \)
Example (constraint-based approach)

```plaintext
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

- \( P_0 \): \([0, 10], [-\infty, \infty], [-\infty, \infty], [-\infty, \infty]\)
- \( P_1 \): \([0, 10], [-10, 100], D_{y_1}^0, D_{y_2}^0\)
- \( P_2 \): \([0, 10], [0, 100], D_{y_1}^0, D_{y_2}^0\)
- \( P_3 \): \([0, 10], [0, 100], [0, 1], D_{y_1}^0\)
- \( P_4 \): \([0, 1], [-1, 0], D_{y_1}^0, D_{y_2}^0\)
- \( P_5 \): \([0, 1], [-1, 0], [2, 3.001], D_{y_2}^0\)
- \( P_6 \): \([0, 10], [0, 100], [0, 1], [-2, -0.667]\)
- \( P_6' \): \([0, 1], [-1, 0], [2, 3.001], [0.666, 2]\)
- \( P_7 \): \([0, 1], [-1, 0], [2, 3.001], [-2, -0.667]\)
Example (constraint-based approach)

```plaintext
float x0 = [0, 10];
float y0 = x0 * x0 - x0;
if (y0 >= 0)
    y1 = x0 / 10;
else
    y1 = x0 * x0 + 2;
y2 = 1 / (y1 - 1.5);
```

```
P0: ([0, 10], [−∞, ∞], [−∞, ∞], [−∞, ∞])
P1: ([0, 10], [−10, 100], D0y1, D0y2)
P2: ([0, 10], [0, 100], D0y1, D0y2)
P3: ([0, 10], [0, 100], [0, 1], D0y1)
P4: ([0, 1], [−1, 0], D0y1, D0y2)
P5: ([0, 1], [−1, 0], [2, 3.001], D0y2)
P6: ([0, 10], [0, 100], [0, 1], [−2, −0.667])
P6': ([0, 1], [−1, 0], [2, 3.001], [0.666, 2])
P7: D7y2 = D6y2 ∪ D6'y2
```
Example (constraint-based approach)

float \( x_0 \) = [0, 10];
float \( y_0 \) = \( x_0 \times x_0 - x_0 \);
if (\( y_0 \) >= 0)
  \( y_1 \) = \( x_0 / 10 \);
else
  \( y_1 \) = \( x_0 \times x_0 + 2 \);
\( y_2 \) = 1 / (\( y_1 - 1.5 \));

\[ y_0 = x_0 \times x_0 - x_0 \]
\[ y_0 \geq 0 \]
\[ y_0 < 0 \]

\[ y_1 = x_0 / 10 \]
\[ y_1 = x_0 \times x_0 + 2 \]
\[ y_2 = 1 / (y_1 - 1.5) \]
\[ y_2 = 1 / (y_1 - 1.5) \]

\( P_0 \): (\([0, 10], [-\infty, \infty], [-\infty, \infty], [-\infty, \infty]\) )
\( P_1 \): (\([0, 10], [-10, 100], D_{y_1}^0, D_{y_2}^0 \) )
\( P_2 \): (\([0, 10], [0, 100], D_{y_1}^0, D_{y_2}^0 \) )
\( P_3 \): (\([0, 10], [0, 100], [0, 1], D_{y_1}^0 \) )
\( P_4 \): (\([0, 1], [-1, 0], D_{y_1}^0, D_{y_2}^0 \) )
\( P_5 \): (\([0, 1], [-1, 0], [2, 3.001], D_{y_2}^0 \) )
\( P_6 \): (\([0, 10], [0, 100], [0, 1], [-2, -0.667] \) )
\( P_6' \): (\([0, 1], [-1, 0], [2, 3.001], [0.666, 2] \) )

\( P_7 \): \( D_{y_2}^7 = [-2, -0.667] \cup [0.666, 2] \)
Example (constraint-based approach)

float \(x_0 = [0, 10]\);
float \(y_0 = x_0 \times x_0 - x_0;\)
if \((y_0 \geq 0)\)
    \(y_1 = x_0 / 10;\)
else
    \(y_1 = x_0 \times x_0 + 2;\)
\(y_2 = 1 / (y_1 - 1.5);\)
Abstract Interpretation, Fluctuat

Static analyzer of C programs using Zonotopes → estimates rounding errors and their propagation

- Programs are considered both:
  - As a specification over the reals
  - As an implementation over the floats

- Fluctuat computes:
  - An over-approximation of the domain-bounds of each variable considered as a real number
  - An over-approximation of the domain-bounds of each variable considered as a floating-point number
  - An over-approximation of the error associated with the variable (difference between floating-point and real number values)
  - The contribution of each instruction to the error
Zonotopes

- **Intuition:** convex polytopes with a central symmetry

Sets of **affine forms** $x = x_0 + x_1 \varepsilon_1 + \cdots + x_n \varepsilon_n$

with $\varepsilon_i \in [-1, 1]$

- **Advantages:**
  - Linear correlations between variables are preserved
  - Nonlinear operations are over-approximated by introducing an error term
  - Good trade-off between performance and precision

- **Limits:**
  - Better than the intervals, not as good as polyhedra
  - Not very accurate for nonlinear terms
  - Not accurate on very common program constructions such as if
Proposed approach

Overview

We use constraint-based local consistencies to reduce the domains of variables computed by Fluctuat.
Proposed approach

Details

• **Set of CSP** generated for a C program:
  • A CSP is built “on the fly” while exploring a path
    Inconsistent CSP $\rightarrow$ current path is cut off
  • Loops are unfolded a finite number of times

• **Filtering:**
  • **Reals:** Hull & Box consistency – RealPaver
  • **Floats:** 3B consistency – FPCS

• Reduced domain of a variable: union of the intervals generated for this variable while filtering **all successful paths** of the program
Experiments

- Programs
  - quadratic: computing the roots of a quadratic equation (GSL library) – conditionals
  - sinus7: expression of the 7th-order Taylor series of function sinus – nonlinearity
  - rump: polynomial of Rump – nonlinearity
  - sqrt: square root computation (Babylonian method) – iterative program

- Expected loss of accuracy of Fluctuat
  - Union at the earliest of program states (join operator): quadratic, sqrt;
  - Domain intersection due to conditional statements (meet operator): quadratic, sqrt;
  - Interpolation by expanding approximations due to loops (widening operator): sqrt;
  - Nonlinear expression approximation: quadratic, sinus7, rump, sqrt.
## Results over the reals

<table>
<thead>
<tr>
<th></th>
<th>Fluctuat (AI)</th>
<th></th>
<th>RealPaver (CP)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Domain</td>
<td>Time</td>
<td>Domain</td>
<td>Time</td>
</tr>
<tr>
<td>quadratic₁ x₀</td>
<td>[−∞, ∞]</td>
<td>0.1 s</td>
<td>[−∞, 0]</td>
<td>1.5 s</td>
</tr>
<tr>
<td>quadratic₁ x₁</td>
<td>[−∞, ∞]</td>
<td>0.1 s</td>
<td>[−8.011, ∞]</td>
<td>1.5 s</td>
</tr>
<tr>
<td>quadratic₂ x₀</td>
<td>[−2e6, 0]</td>
<td>0.1 s</td>
<td>[−1e6, 0]</td>
<td>0.5 s</td>
</tr>
<tr>
<td>quadratic₂ x₁</td>
<td>[−1e6, 0]</td>
<td>0.1 s</td>
<td>[−5.186e5, 0]</td>
<td>0.5 s</td>
</tr>
<tr>
<td>sinus7</td>
<td>[−1.009, 1.009]</td>
<td>0.1 s</td>
<td>[−0.842, 0.843]</td>
<td>0.3 s</td>
</tr>
<tr>
<td>rump</td>
<td>[−1e37, 2e37]</td>
<td>0.1 s</td>
<td>[−1e36, 1.7e37]</td>
<td>1.2 s</td>
</tr>
<tr>
<td>sqrt₁</td>
<td>[2.116, 2.354]</td>
<td>0.1 s</td>
<td>[2.121, 2.346]</td>
<td>0.3 s</td>
</tr>
<tr>
<td>sqrt₂</td>
<td>[2.098, 3.435]</td>
<td>0.1 s</td>
<td>[2.232, 3.165]</td>
<td>0.5 s</td>
</tr>
</tbody>
</table>
FPCS

Correct solver over the floats based on 2B-consistency: no solution lost

- **Projection functions** for floats:
  - Direct projection: straightforward adaptation of interval arithmetic
  - Inverse projection: uses a larger format than the system variables

- Handling of **rounding modes, nonlinear expressions** and the usual **mathematical functions** (trigonometric...)
- Handling of x86 architecture specifics
## Results over the floats

<table>
<thead>
<tr>
<th></th>
<th>Fluctuat (AI)</th>
<th>FPCS (CP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Domain</td>
<td>Time</td>
</tr>
<tr>
<td>quadratic(_1) (x_0)</td>
<td>([-\infty, \infty])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>quadratic(_1) (x_1)</td>
<td>([-\infty, \infty])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>quadratic(_2) (x_0)</td>
<td>([-2e6, 0])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>quadratic(_2) (x_1)</td>
<td>([-1e6, 0])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>sinus(_7)</td>
<td>([-1.009, 1.009])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>rump</td>
<td>([-1e37, 2e37])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>sqrt(_1)</td>
<td>([2.116, 2.354])</td>
<td>0.1 s</td>
</tr>
<tr>
<td>sqrt(_2)</td>
<td>([-\infty, \infty])</td>
<td>0.1 s</td>
</tr>
</tbody>
</table>
Conclusion

• CP
  • Advantages:
    → Good refutation capabilities
    → Handling nonlinear constraints
  • Limits: Distinct exploration of each executable path is a critical issue

• AI
  • Advantages:
    → Good scaling capabilities
    → Zonotopes are better approximations of linear constraints than boxes
  • Limits: Over-approximations can be very rough

→ Complementary techniques → hybrid approach
• Greater domain reduction achieved when combining both approaches
• Automated and tight cooperation is promising
Abstract domain intersection

```c
1/* Pre-condition : x ∈ [0,10] */
2double conditional(double x) {
3    double y = x*x - x;
4    if (y >= 0)
5        y = x/10;
6    else
7        y = x*x + 2;
8    return y;
9}
```
Quadratic equation roots

```c
int quadratic(double a, double b, double c) {
    double r, sgnb, temp, r1, r2
    double disc = b * b - 4 * a * c;
    if (a == 0) {
        if (b == 0)
            return 0;
        else {
            x0 = -c / b;
            return 1;
        }
    }
    if (disc > 0) {
        if (b == 0) {
            r = fabs (0.5 * sqrt (disc) / a);
            x0 = -r;
            x1 = r;
        } else {
            sgnb = (b > 0 ? 1 : -1);
            temp = -0.5 * (b + sgnb * sqrt (disc));
            r1 = temp / a;
            r2 = c / temp;
            if (r1 < r2) {
                x0 = r1;
                x1 = r2;
            } else {
                x0 = r2;
                x1 = r1;
            }
        }
    } else if (disc == 0) {
        x0 = -0.5 * b / a;
        x1 = -0.5 * b / a;
        return 2;
    } else
    return 0;
}
```
7th-order Taylor series of function \textit{sinus}

```c
double sinus(double x)
{ return x - x*x*x/6 + x*x*x*x*x/120 + x*x*x*x*x*x*x*x/5040; }
```

\textbf{Rump’s polynomial}

```c
double rump(double x, double y) {
    double f;
    f = 333.75*y*y*y*y*y*y;
    f = f + x*x*(11*x*x*y*y - y*y*y*y*y*y - 121*y*y*y*y - 2);
    f = f + 5.5*y*y*y*y*y*y*y*y;
    f = f + x / (2*y);
    return f;
}
```

\textbf{Square root function}

```c
double sqrt(double x) {
    double xn, xn1;
    xn = x/2;
    xn1 = 0.5*(xn + x/xn);
    while (xn-xn1 > 1e-2) {
        xn = xn1;
        xn1 = 0.5*(xn + x/xn);
    }
    return xn1;
}
```