Refining AI Analysis with CP Techniques
or
How to identifying suspicious values in programs with floating-point numbers

Michel RUEHER

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

(joined work with Olivier Ponsini, Claude Michel)

JFPC

June 2013
Introduction

• **Problem:** verifying programs with **floating-point computations**

Embedded systems written in C (transportation, nuclear plants,...)

• **Programs use floating-point numbers** but
  
  ▶ Specifications are written with the *semantics of reals* “in mind”
  
  ▶ Programs are written with the *semantics of reals* “in mind”
Floating-point arithmetic pitfalls

Rounding $\leadsto$ Counter-intuitive properties

$$(0.1)_{10} = (0.000110011001100\cdots)_2$$

simple precision $\leadsto$ 0.10000001490116119384765625

- Neither associative nor distributive operators
  $$(−10000001 + 10^7) + 0.5 \neq −10000001 + (10^7 + 0.5)$$

- Absorption, cancellation phenomena
  Absorption: $10^7 + 0.5 = 10^7$
  Cancellation: $((1 − 10^{-7}) − 1) * 10^7 = −1.192\cdots(\neq −1)$

$\rightarrow$ Floats are source of errors in programs
Objectives & Method

**Goals:**

→ bounds for variables with real numbers semantics and floating-point numbers semantics
→ bounds for the error due to the use of floating-point numbers instead of real numbers

~ to identify suspicious values

**Method:** combining *abstract interpretation* & *constraint programming*
Outline

Problematic: Verifying Programs with FP computations

AI Approach: Abstraction of program states

Constraint Programming over continuous domains

Example 1

Combining AI and CP

Experiments

Conclusion
AI Approach: Abstraction of program states

Intervals, zonotopes, polyhedra...

Zonotopes: convex polytopes with a central symmetry
Sets of affine forms
\[ \hat{a} = a_0 + a_1 \varepsilon_1 + \cdots + a_n \varepsilon_n \]
\[ \hat{b} = b_0 + b_1 \varepsilon_1 + \cdots + b_n \varepsilon_n \]
with \( \varepsilon_i \in [-1, 1] \)

+ Good trade-off between performance and precision
– Not very accurate for nonlinear expressions
– Not accurate on very common program constructs such as conditionals
AI: Static analysis (cont.)

+ **Good scalability** for
  - Showing absence of runtime errors
  - Estimating rounding errors and their propagation
  - Checking properties of programs

– **Lack of precision**
  - Approximations may be very coarse
  - Over-approximation \(\leadsto\) possible false alarms
AI & False alarm

From Cousot:

http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html
CP over continuous domains
A branch & prune process

Iteration of two steps:

1. **Pruning the search space**
2. **Making a choice to generate two (or more) sub-problems**

Pruning step → **reduces an interval** when the upper bound or the lower bound does not satisfy some constraint
Branching step → **splits the domain** of some variable in two or more intervals
Local consistencies – 2B–consistency

- A constraint $c_j$ is 2B–consistent if for any variable $x_i$ of $c_j$, the bounds $D_{x_i}$ and $\overline{D}_{x_i}$ have a support in the domains of all other variables of $c_j$.

  → Variable $x$ is 2B–consistent for $f(x, x_1, \ldots, x_n) = 0$ if the lower (resp. upper) bound of the domain of $x$ is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$.

A CSP is 2B–consistent iff all its constraints are 2B–consistent.
3B–Consistency (1)

3B–Consistency, a shaving process

→

checks whether 2B–Consistency can be enforced when the domain of a variable is reduced to the value of one of its bounds in the whole system
Constraint Programming framework: sum up

+ Good **refutation** capabilities
  **Flexibility**: handling of integers, floats, non-linear expressions,...

− **Scalability**
  Pruning may be costly for **large domains**
  A CSP is a conjunction of constraints \(\sim\) a different constraint system is required for each path of the CFG
Example 1

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

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

$P_0: \hat{x}^0 = 5 + 5\varepsilon_1 \quad \varepsilon_1 \in [-1, 1]$
$D_x^0 = [0, 10]$

$P_1: \hat{y}^1 = 32.5 + 45\varepsilon_1 + 12.5\eta_1$
$\eta_1 \in [-1, 1]$
$D_x^1 = [0, 10] \quad D_y^1 = [-10, 90]$

$P_2: \hat{y}^2 = \hat{y}^1 \quad D_x^2 = [0, 10]$
$D_y^2 = [0, 90]$

$P_3: \hat{y}^3 = 0.5 + 0.5\varepsilon_1$
$D_y^3 = [0, 1]$

$P_4$

$P_5$

$P_6$
Example 1: Abstract Interpretation (zonotopes)

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

- **Example 1**: Abstract Interpretation (zonotopes)
  - Initial state: $x^0 = 5 + 5\varepsilon_1$, $\varepsilon_1 \in [-1, 1]$
  - Domain: $D_x^0 = [0, 10]$

- **Path 1** ($P_0$)
  - Condition: $y \geq 0$
  - Transition: $y = x/10$
  - Domain: $D_x^1 = [0, 10]$
  - Domain of y: $D_y^1 = [-10, 90]$

- **Path 2** ($P_2$)
  - Condition: $y \geq 0$
  - Transition: $y = x/10$
  - Domain: $D_x^2 = [0, 10]$
  - Domain of y: $D_y^2 = [0, 90]$

- **Path 3** ($P_3$)
  - Transition: $y = 0.5 + 0.5\varepsilon_1$
  - Domain: $D_y^3 = [0, 1]$

- **Path 4** ($P_4$)
  - Condition: $y < 0$
  - Transition: $y = x*x + 2$
  - Domain: $D_x^4 = [0, 10]$
  - Domain of y: $D_y^4 = [-10, 0]\]

- **Path 5** ($P_5$)
  - Transition: $y = x*x + 2$

- **Path 6** ($P_6$)
Example 1: Abstract Interpretation (zonotopes)

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

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

\[ P_0: \hat{x}^0 = 5 + 5\varepsilon_1 \quad \varepsilon_1 \in [-1, 1] \]
\[ D_x^0 = [0, 10] \]

\[ y \geq 0 \]

\[ P_1: \hat{y}^1 = 32.5 + 45\varepsilon_1 + 12.5\eta_1 \]
\[ \eta_1 \in [-1, 1] \]
\[ D_x^1 = [0, 10] \quad D_y^1 = [-10, 90] \]

\[ y < 0 \]

\[ P_2: \hat{y}^2 = \hat{y}^1 \quad D_x^2 = [0, 10] \]
\[ D_y^2 = [0, 90] \]

\[ P_3: \hat{y}^3 = 0.5 + 0.5\varepsilon_1 \]
\[ D_y^3 = [0, 1] \]

\[ y = x/10 \]

\[ \hat{y}^4 = \hat{y}^1 \quad D_x^4 = [0, 10] \]
\[ D_y^4 = [-10, 0] \]

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

\[ P_5: \hat{y}^5 = 39.5 + 50\varepsilon_1 + 12.5\eta_1 \]
\[ \eta_1 \in [-1, 1] \]
\[ D_x^5 = [2, 102] \]

\[ P_6: \hat{y}^6 = \hat{y}^3 \cup \hat{y}^5 = 39.5 + 0.5\varepsilon_1 + 62\eta_2 \]
\[ \eta_2 \in [-1, 1] \]
\[ D_y^6 = D_y^3 \cup D_y^5 = [0, 102] \]
Example 1: Constraint Programming

\[ P_0: D_{x_0} = [0, 10] \quad D_{y_0} = [-10, 90] \quad D_{y_1} = [0, 102] \]

\[
\begin{align*}
y_0 &= x_0 \times x_0 - x_0 \\
y_0 &\geq 0 \\
y_1 &= x_0 / 10 \\
\end{align*}
\]

Filtering:

\[
\begin{align*}
D_{x_0}^1 &= [0, 10] \\
D_{y_0}^1 &= [0, 90] \\
D_{y_1}^1 &= [0, 1] \\
\end{align*}
\]
Example 1: Constraint Programming

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

\[ y_0 \geq 0 \]

\[ y_1 = x_0 / 10 \]

\[ D_{x_0} = [0, 10] \]

\[ D_{y_0} = [-10, 90] \]

\[ D_{y_1} = [0, 102] \]

\[ P_0: \]

\[ P_6: \]

\[ D_{x_0}^2 = [0, 1.026] \]

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

\[ y_0 < 0 \]

\[ y_1 = x_0 \times x_0 + 2 \]

filtering
Example 1: Constraint Programming

\[ P_0: D_{x_0} = [0, 10] \quad D_{y_0} = [-10, 90] \quad D_{y_1} = [0, 102] \]

\[ y_0 = x_0 \times x_0 - x_0 \]
\[ y_0 \geq 0 \]
\[ y_1 = x_0 / 10 \]

Filtering

\[ D_{x_0}^1 = [0, 10] \]
\[ D_{y_0}^1 = [0, 90] \]
\[ D_{y_1}^1 = [0, 1] \]

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

\[ D_{x_0}^2 = [0, 1.026] \]
\[ D_{y_0}^2 = [-0.257, 0] \]
\[ D_{y_1}^2 = [2, 3.027] \]

\[ P_6: D_{y_1}^3 = D_{y_1}^1 \cup D_{y_1}^2 = [0, 3.027] \]
Proposed approach: Combining AI and CP

Successive exploration and merging steps

• Use of AI to compute a *first approximation* of the values of variables at a program node where two branches join

• Building a constraint system for each branch between two join nodes in the CFG of the program and use of CP local consistencies *to shrink the domains* computed by AI
Filtering techniques

- **FPCS**: 3B(w)-consistency over the floats
  - Projection functions for floats
  - Handling of rounding modes
  - Handling of x86 architecture specifics

- **RealPaver**: 2B(w)-consistency & Box-consistency over the reals
  - Reliable approximations of continuous solution sets
  - Correctly rounded interval methods and constraint satisfaction techniques
Experiments: benchmarks

• Illustrative programs
  ▶ **quadratic** → real roots of a quadratic equation (GNU scientific library); contains many conditionals
  ▶ **sinus7** → the 7th-order Taylor series of function sinus
  ▶ **sqrt** → an approximate value (error of $10^{-2}$) of the square root of a number greater than 4 (Babylonian method)
  ▶ **bigLoop**: contains non-linear expressions followed by a loop that iterates one million times
  ▶ **rump**: a very particular polynomial designed to outline a catastrophic cancellation phenomenon

• 55 benchs from CDFL, a program analyzer for proving the absence of runtime errors in program with floating-point computations based on *Conflict-Driven Learning*
Experiments: Results over the floating-point numbers

<table>
<thead>
<tr>
<th>Fluctuat (AI)</th>
<th>RAICP (AI + CP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Time</td>
</tr>
<tr>
<td>quadratic ( x_0 )</td>
<td>([-\infty, \infty])</td>
</tr>
<tr>
<td>quadratic ( x_1 )</td>
<td>([-\infty, \infty])</td>
</tr>
<tr>
<td>quadratic ( x_0 )</td>
<td>([-2e6, 0])</td>
</tr>
<tr>
<td>quadratic ( x_1 )</td>
<td>([-1e6, 0])</td>
</tr>
<tr>
<td>sinus7</td>
<td>([-1.009, 1.009])</td>
</tr>
<tr>
<td>rump</td>
<td>([-1.2e37, 2e37])</td>
</tr>
<tr>
<td>sqrt1</td>
<td>([2.116, 2.354])</td>
</tr>
<tr>
<td>sqrt2</td>
<td>([-\infty, \infty])</td>
</tr>
<tr>
<td>bigLoop</td>
<td>([-\infty, \infty])</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

**Fluctuat**: state-of-the-art AI analyzer for estimating rounding errors and their propagation using zonotopes
Experiments: Results over the real numbers

<table>
<thead>
<tr>
<th></th>
<th>Fluctuat (AI)</th>
<th>RAiCP (AI + 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>0.14 s</td>
</tr>
<tr>
<td>quadratic$_1$ $x_1$</td>
<td>$[-\infty, \infty]$</td>
<td>0.14 s</td>
</tr>
<tr>
<td>quadratic$_2$ $x_0$</td>
<td>$[-2e6, 0]$</td>
<td>0.14 s</td>
</tr>
<tr>
<td>quadratic$_2$ $x_1$</td>
<td>$[-1e6, 0]$</td>
<td>0.14 s</td>
</tr>
<tr>
<td>sinus7</td>
<td>$[-1.009, 1.009]$</td>
<td>0.12 s</td>
</tr>
<tr>
<td>rump</td>
<td>$[-1.2e37, 2e37]$</td>
<td>0.13 s</td>
</tr>
<tr>
<td>sqrt$_1$</td>
<td>$[2.116, 2.354]$</td>
<td>0.13 s</td>
</tr>
<tr>
<td>sqrt$_2$</td>
<td>$[2.098, 3.435]$</td>
<td>0.2 s</td>
</tr>
<tr>
<td>bigLoop</td>
<td>$[-\infty, \infty]$</td>
<td>0.15 s</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1.29 s</strong></td>
<td></td>
</tr>
</tbody>
</table>
Experiments: eliminating false alarms

**CDFL:** Program analyzer for proving the absence of runtime errors in program with floating-point computations based on *Conflict-Driven Learning*

<table>
<thead>
<tr>
<th></th>
<th>RAiCP</th>
<th>Fluctuat</th>
<th>CDFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>False alarms</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Total time</td>
<td>40.55 s</td>
<td>18.37 s</td>
<td>208.99 s</td>
</tr>
</tbody>
</table>

Computed on the 55 benches from CDFL paper (TACAS’12, D’Silva, Leopold Haller, Daniel Kroening, Michael Tautschnig)
Conclusion

**AI + CP framework:** Efficient computation and sharp good domain approximations

**Further works:** interact with AI at the abstract domain level
  - Better approximations
  - Keep statement contribution to rounding errors