Symbolic Path-Oriented Test Data Generation for Floating-Point Programs

Arnaud Gotlieb
Certus V&V Centre,
SIMULA Research. Lab.,
Norway

Joint work with Roberto Bagnara (Parma), Matthieu Carlier (IRISA) and Roberta Gori (Pisa)
Motivations

- Increasing use of *floating-point computations* in safety-critical systems
- Testing for detecting and evaluating *rounding errors*
- Focus on program paths that expose the system to these errors
Symbolic execution of floating-point computations

- **Symbolic Execution** is a popular technique in automatic test input generation (e.g., PathCrawler, PEX, SAGE, KLEE, ...)

  path $\rightarrow$ path conditions $\rightarrow$ constraint solving $\rightarrow$ test input

- However, handling **correctly** floating-point computations in constraint solving is difficult
```c
float foo(float x) {
    float y = 1.0e12
    if (x < 10000.0) {
        z = x + y
        if (z > y)
            ...
    }
}
```

Is the path 1-2-3-4 feasible?

Path conditions:
- \(x < 10000.0\)
- \(x + 1.0e12 > 1.0e12\)

On the reals: \(x \in (0,10000)\)

On the floats: no solution!
Conversely,

```c
float foo( float x ) {
    float y = 1.0e12
    1. if( x > 0.0 )
    2. z = x + y
    3. if( z == y)
    4. ...
}
```

Is the path 1-2-3-4 feasible?

Path conditions:
- \( x > 0.0 \)
- \( x + 1.0e12 = 1.0e12 \)

On the reals: no solution!

On the floats: \( x \in (0, 32767.99...) \)
Contributions of the talk

- Understanding rounding errors and why they occur in numerical programs
- How to solve a set of floating-point constraints
- Claim: symbolic path-oriented test input generation for floating-point programs is feasible!
Outline

• IEEE-754 and rounding errors

• Constraint solving over the floats

• FPSE and first experimental results

• Conclusions
Binary floating-point numbers (IEEE-754)

- float: \((s, f, e)\) a bit pattern of 32, 64 or more bits

\[ 0 < e < e_{\text{max}}: \text{Normalized} \]

\[ (-1)^s \ 1.f \ 2^{(e - \text{bias})} \]

- Significand (23, 52 bits or extended)
- Exponent (8, 11 bits or extended)

- \(e = 0:\) Denormalized \((-1)^s \ 0.f \ 2^{(-\text{bias} + 1)}\)
  
  +0.0, -0.0

- \(e = e_{\text{max}}:\) +INF, -INF, NaNs

- Rounding: \(r(\ '1.0e12'\ ) = 999999995904.0_f\)

4 modes (near-to-even, ...), monotonicity (i.e., if \(x > y\) then \(r(x) > r(y)\))
Accurcy requirement of IEEE-754

For add, sub, mul, div, sqr, rem, conv:
the floating-point result of an operation must be the rounding result of the exact operation over the reals

‘1.0e12’ add ‘10000.0’

= \text{r}(‘1.0e12’) \text{ add } \text{r}(‘10000.0’)
= 999999995904.0_f \text{ add } 10000.0_f
= \text{r}(999999995904.0_f + 10000.0_f)
= \text{r}(‘1000000005904.0’)
= 999999995904.0_f
= ‘1.0e12’

Poor (but well-conceived) approximation of the reals

\[ 999999995904.0_f + 10000.0_f \]
Decomposition in symbolic execution

- Decomposition in SSA-like three-address code, preserving evaluation order
  
e.g., \( z := z \ast z + z \)  \( \Rightarrow \)  \( t1 == z1 \text{ mul } z1, \ z2 == t1 \text{ add } z1 \)

- Temporary results are stored into known formats
  (requires to set up specific options when compiling)
Outline

• IEEE-754 and rounding errors

• Constraint solving over the floats

• FPSE and first experimental results

• Conclusions
Context of this work

- Programs that strictly conform to IEEE-754
  
  \[ E ::= E \text{ add } E \mid E \text{ subs } E \mid E \text{ mult } E \mid E \text{ div } E \mid E == E \mid E != E \mid E > E \mid E >= E \mid (\text{float}) E \mid (\text{double}) E \mid \text{Var} \mid \text{Constants} \]

- No extended-formats, only the to-the-nearest rounding mode, no exception, no NaNs

- Decomposition preserves the order of evaluation

- Temporary results are stored in known formats (requires to set up specific options when compiling)
Simple Symbolic Execution

Notations: Control Flow Graph \((N, A, e, s)\)
- \(X\) vector of symbolic input

Definition (Symbolic State):
\((\text{Path, State, PC})\) where

- \(\text{Path} = n_i \rightarrow \ldots \rightarrow n_j\) is a (partial) path of the CFG
- \(\text{State} = \{<v, \varphi>\}_{v \in \text{Var}(P)}\) \(\varphi\) is an algebraic expr. over \(X\)
- \(\text{PC} = c_1, \ldots, c_n\) a finite conjunction of conditions over \(X\) or a temporary assignments
(Path, State, PC) : examples

(1, \{<x, X>, <y, 1.0e12>, <z, \bot>\}, true)

(1 → 2 → 3,
\{<x, X>, <y, 1.0e12>, <z, X+1.0e12>\}, 
X < 10000.0)

(1 → 2 → 3 → 4,
\{<x, X>, <y, 1.0e12>, <z, X+1.0e12>\}, 
X<10000.0, T := X add 1.0e12, T > 1.0e12)

float foo(float x) {
    float y = 1.0e12;
    if (x < 10000.0) { 
        z = x + y;
    } 
    if (z > y) {
        ... 
    }
    ... 
}
Symbolic state: features

- \((\text{Path}, \text{State}, \text{PC})\) is computed either by a forward or a backward analysis over the vertex of \text{Path}.

- Let \(S_{\text{PC}}\) be the solution-set of \text{PC}.
  Then \(\forall X \in S_{\text{PC}}, \text{Path} \text{ is activated by } X\).

- When \(S_{\text{PC}} = \emptyset\) then \text{Path} is non-feasible.

However, finding all the non-feasible paths is a classical undecidable problem [Weyuker 79].
Interval propagation

- Var $x$ abstracted by an interval $I_x$

- Interval Arithmetic:

  $I_x = [a,b]$ and $I_y = [c,d]$ then $I_{x+y} = [r(a+c), r(b+d)]$

  $I_{x-y} = [r(a-d), r(b-c)]$

  $I_{\exp(x)} = [r(\exp(a)), r(\exp(b))]$ ...

- Filtering over intervals using projection functions

  \[
  \begin{cases}
  I_z' \leftarrow I_{x+y} \cap I_z \\
  [z = x + y] \quad \text{leads to} \quad \begin{cases}
  I_x' \leftarrow I_{z-y} \cap I_x \\
  I_y' \leftarrow I_{z-x} \cap I_y
  \end{cases}
  \end{cases}
  \]

  Filtering, constraint propagation and labelling $\Rightarrow$ constraint solving
Example: \( y = \log(x), \ x + y = 0 \)

4 projection functions

\[
\begin{align*}
I_x' & \leftarrow I_{\exp(y)} \cup I_x \quad \text{(1)} \\
I_y' & \leftarrow I_{\log(x)} \cup I_y \quad \text{(2)} \\
I_x' & \leftarrow I_{-y} \cup I_x \quad \text{(3)} \\
I_y' & \leftarrow I_{-x} \cup I_y \quad \text{(4)}
\end{align*}
\]

\( x \in [-\infty, +\infty] \quad [0, +\infty] \quad [0, 1] \quad [0.56, 1] \quad [0.56, 0.57] \)

\( y \in [-\infty, +\infty] \quad [-\infty, 0] \quad [-1, 0] \quad [-1, -0.56] \quad [-0.57, -0.56] \)

If there is a solution \( x \), then \( x \in [0.56, 0.57] \)

True over the reals, can be adapted for floating-point numbers!

Solving constraints means also detecting unsatisfiability
Existing solvers based on IP

Over the reals:

- INTERLOG (Botella & Taillibert 1993, Lhomme 1993)
  Dynamic optimizations (Lhomme Gotlieb Rueher 1996)
- NUMERICA (Van Hentenryck 1997)
- REALPAVER (Granvilliers 1998)

Over the floats:

- FPCS (Michel Rueher Lebbah 2001)
- FPSE (Botella Gotlieb Michel 2006)
- ECLAIR (Bagnara et al. BUGSENG 2011)
Our approach to solve path conditions: Interval propagation over floating-point variables

- Notations:

Recall that \([a \text{ add } b]\) denotes \(\text{near}(a + b)\)

- Path conditions are made of constraints and assignments
Our approach: floating-point projections

Direct and indirect projections for the assignment:

\[ \text{proj}(r, r := a \text{ add } b) \quad \text{(direct)} \]
\[ [r := a \text{ add } b] \quad \text{leads to} \quad \text{proj}(a, r := a \text{ add } b) \quad \text{(1st indirect)} \]
\[ \text{proj}(b, r := a \text{ add } b) \quad \text{(2nd indirect)} \]

Direct projections (over numeric fp numbers):

If \( l_r = [r_l, r_h] \), \( l_a = [a_l, a_h] \) and \( l_b = [b_l, b_h] \) then

\[ [r := a \text{ add } b] \quad [r_l', r_h'] \leftarrow [a_l \text{ add } b_l, a_h \text{ add } b_h] \cap [r_l, r_h] \]
\[ [r := a \text{ subs } b] \quad [r_l', r_h'] \leftarrow [a_l \text{ subs } b_h, a_h \text{ subs } b_l] \cap [r_l, r_h] \]

...
Ex: Direct projection \[ r := a \text{ add } b \]

Monotony of rounding: 
\[ r_1 \leq r_2 \Rightarrow \text{near}(r_1) \leq \text{near}(r_2) \]
More complex: indirect projections

If $I_r = [r_l, r_h], \ I_a = [a_l, a_h]$ and $I_b = [b_l, b_h]$ then

1st indirect projection of $[r := a \text{ add } b]$
$[a'_l, a'_h] \leftarrow [\text{mid}(r_l, r_l^-) \text{ subs } b_h, \text{mid}(r_h, r_h^+) \text{ subs } b_l] \cap [a_l, a_h]$

1st indirect projection of $[r := a \text{ subs } b]$
$[a'_l, a'_h] \leftarrow [\text{mid}(r_l, r_l^-) \text{ add } b_l, \text{mid}(r_h, r_h^+) \text{ add } b_h] \cap [a_l, a_h]$

2nd indirect projection of $[r := a \text{ subs } b]$
$[b'_l, b'_h] \leftarrow [a_l \text{ subs } \text{mid}(r_h, r_h^+), a_h \text{ subs } \text{mid}(r_l, r_l^-)] \cap [b_l, b_h]$
Ex: 1\textsuperscript{st} indirect projection \[ r := a \text{ add } b \]

\[ a_h' \leftarrow \min(\text{mid}(r_h, r_{h^+}) \text{ subs } b_l, a_h) \]

\[ a_l' \leftarrow \max(\text{mid}(r_l, r_{l^-}) \text{ subs } b_h, a_l) \]

impossible

not optimal, but computable with to-the-nearest
Handling comparisons and conversions

Comparisons (1st proj):

\[ [a'_l, a'_h] \leftarrow [\max(a_l, b_l), \min(a_h, b_h)] \]
when \[ a == b \]

\[ [a'_l, a'_h] \leftarrow [\max(a_l, b_l)^+, a_h] \]
when \[ a > b \]

\[ [a'_l, a'_h] \leftarrow [\text{if}(a_l = b_l = b_h) \text{ then } a_l^+ \text{ else } a_l, \text{ if}(a_h = b_l = b_h) \text{ then } a_h^- \text{ else } a_h] \]
when \[ a != b \]

Floating-point conversions:

when \[ [r := (\text{float})a] \]

\[ [r'_l, r'_h] \leftarrow [\max_f((\text{float})a_l, r_l), \min_f((\text{float})a_h, r_h)] \]
(direct proj.)

\[ [a'_l, a'_h] \leftarrow [\max_d(a_l, \text{mid}(r_l, r_l^-)), \min_d(a_h, \text{mid}(r_h, r_h^+))] \]
(indirect)
Handling zeros and infinities

Based on an extended arithmetic defined by specific tables:

Values of $a$ in 1st indirect projection of $[r := a \text{ add } b]$

<table>
<thead>
<tr>
<th>$b \setminus r$</th>
<th>-INF</th>
<th>-0.0</th>
<th>+0.0</th>
<th>Nv</th>
<th>+INF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-INF</td>
<td>Nv,</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>-INF</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>-0.0</td>
<td>-INF</td>
<td>-0.0</td>
<td>+0.0</td>
<td>Nv</td>
<td>+INF</td>
</tr>
<tr>
<td>+0.0</td>
<td>-INF</td>
<td>--</td>
<td>±0.0</td>
<td>Nv</td>
<td>+INF</td>
</tr>
<tr>
<td>Nv</td>
<td>Nv, -INF</td>
<td>--</td>
<td>Nv,±0.0</td>
<td>Nv,±0.0</td>
<td>Nv,+INF</td>
</tr>
<tr>
<td>+INF</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Nv,+INF,±0.0</td>
</tr>
</tbody>
</table>
The Marre & Michel property (Marre and Michel 2010)

\[ X - Y = a \]

Then, the property says that \( Y \) cannot be greater than \( b \).

1. We have reformulated and corrected this property \( \rightarrow \) ULP-Maximum
Filtering by ULP-Maximum

2. And we have generalized it to **mul** and **div**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>( x \subseteq \cdot )</th>
<th>( y \subseteq \cdot )</th>
<th>Condition(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z = x \oplus y, \ 0 &lt; z \leq f_{\text{max}} )</td>
<td>([ \delta_\oplus(\zeta), \overline{\delta_\oplus(\zeta')} ])</td>
<td>([ \delta_\oplus(\zeta), \overline{\delta_\oplus(\zeta')} ])</td>
<td>( \zeta = \mu_\oplus(z), \ -f_{\text{max}} \leq \delta_\oplus(\zeta), \ \overline{\delta_\oplus(\zeta')} \leq f_{\text{max}} )</td>
</tr>
<tr>
<td>( z = x \oplus y, \ -f_{\text{max}} \leq z &lt; 0 )</td>
<td>([ -\delta_\oplus(\zeta'), -\delta_\oplus(\zeta') ])</td>
<td>([ -\overline{\delta_\oplus(\zeta')}, -\delta_\oplus(\zeta') ])</td>
<td>( \zeta' = \mu_\oplus(-z), \ -f_{\text{max}} \leq \delta_\oplus(\zeta'), \ \overline{\delta_\oplus(\zeta')} \leq f_{\text{max}} )</td>
</tr>
<tr>
<td>( z = x \ominus y, \ 0 &lt; z \leq f_{\text{max}} )</td>
<td>([ \delta_\ominus(\zeta), \overline{\delta_\ominus(\zeta)} ])</td>
<td>([ \delta_\ominus(\zeta'), \overline{\delta_\ominus(\zeta)} ])</td>
<td>( \zeta = \mu_\ominus(z), \ -f_{\text{max}} \leq \delta_\ominus(\zeta), \ \overline{\delta_\ominus(\zeta)} \leq f_{\text{max}} )</td>
</tr>
<tr>
<td>( z = x \ominus y, \ -f_{\text{max}} \leq z &lt; 0 )</td>
<td>([ -\delta_\ominus(\zeta'), -\delta_\ominus(\zeta') ])</td>
<td>([ -\overline{\delta_\ominus(\zeta')}, -\delta_\ominus(\zeta') ])</td>
<td>( \zeta' = \mu_\ominus(-z), \ -f_{\text{max}} \leq \delta_\ominus(\zeta'), \ \overline{\delta_\ominus(\zeta')} \leq f_{\text{max}} )</td>
</tr>
<tr>
<td>( z = x \otimes y, \ 0 &lt;</td>
<td>z</td>
<td>\leq 2(2 - 2^{1-p}) )</td>
<td>([ \delta_\otimes(m), \overline{\delta_\otimes(m)} ])</td>
</tr>
<tr>
<td>( z = x \ominus y, \ 0 &lt;</td>
<td>z</td>
<td>\leq 1 )</td>
<td>([ \delta_\ominus(m), \overline{\delta_\ominus(m)} ])</td>
</tr>
</tbody>
</table>

\[
\overline{\delta_\oplus(z)} = \begin{cases} \beta, & \text{if } 0 < z < +\infty, \\ \alpha, & \text{if } -\infty < z < 0; \end{cases} \qquad \delta_\oplus(z) = -\overline{\delta_\oplus(-z)}; \\
\overline{\delta_\ominus(z)} = |z| \cdot 2^{-e_{\text{min}}}; \qquad \delta_\ominus(z) = -\overline{\delta_\ominus(z)}; \\
\overline{\delta_\otimes(z)} = |z| \otimes f_{\text{max}}; \qquad \delta_\otimes(z) = -\overline{\delta_\otimes(z)}. 
\]

- All the details and correction proofs are in the paper!
Outline

• IEEE-754 and rounding errors
• Constraint solving over the floats
• FPSE and first experimental results
• Conclusions
FPSE: Floating-Point Symbolic Execution

- Handles ISO-C computations on Sparc/Solaris/gcc and Intel/WinXP/VisualC++

Programs that strictly conform to IEEE-754

\[ E ::= E \text{ add } E \mid E \text{ sub } E \mid E \text{ mul } E \mid E \text{ div } E \]
\[ \mid E == E \mid E != E \mid E > E \mid E >= E \]
\[ \mid \text{(float) } E \mid \text{(double) } E \mid \text{ Var } \mid \text{ Constants} \]

- Only near-to-even rounding mode, only normalized numbers

- Written in SICStus Prolog (constraint propagation engine, ~10 KLOC) and C (floating-point projection functions, ~1 KLOC)
An example

```c
/* double-error.c */

int main () {
    double x;
    float y,z,r;
    x=1125899973951488.0;
    y = x + 1;
    z = x - 1;
    r = y - z;
    printf("%f\n", r);
}
```

% 134217728.000000
## Selected experimental results (gcc/solaris/sparc)

<table>
<thead>
<tr>
<th>Programs</th>
<th>Expected results</th>
<th>Eclipse</th>
<th>FPSE</th>
</tr>
</thead>
</table>
| **[Goldberg 91]**  
\(2.0\times10^{-30} + 1.0\times10^{-30} - 1.0\times10^{-30} - 1.0\times10^{-30}\) | single: -1.0000000003e-30  
double: -1.0e-30 | clpr: +0.0,  
clpq: +10^{30} | single: -1.0000000003e-30  
double: -1.0e-30 |
| **[Goldberg 91]**  
\(D = B^2 - 4AC\)  
\(A = 1.22, B = 3.34, D = +0.0\) | single:  
2.2859836065573771  
double: 2.2859836065573770 | clpr: 2.2859836065573771  
clpq: 27889/12200=2.285...  
ic: [2.2859836065573766, 2.2859839065573771] | single:  
2.2859833240509033,  
2.2859836065573770 |
| **X < 1.0e4,  
\(T_1 = X + 1.0e12,  
T_2 > 1.0e12\)** | single: infeasible path  
double: [6.103e-5, 9.999e3] | clpr: (-0.0, 10000.0)  
clpq: (0, 10000)  
ic: [0.0, 10000.0] | single: infeasible path  
double: [6.103e-5, 9.999e3] |
| **X > 0,  
\(T_1 = X + 1.0e12,  
T_2 = X + 1.0e12\)** | single:  
[1.4012984643248171e-45, 3.2767998046875000e+04]  
double:  
[4.94065645841247e-324, 6.1035156250000000e-05] | clpr,clpq: infeasible  
ic: infeasible | single:  
[1.4012984643248171e-45, 3.2768000000000000e+04]  
double:  
[4.94065645841247e-324, 6.1035156250000000e-05] |
| **power.c (X=10, Y = -40)**  
84 constraints | single: +0.0  
double: 1.0000000000001e-40 | clpr: +0.0,  
clpq: +10^{-40}  
ic: [9.99999e-41, 1.0000000e-40] | single: +0.0  
double: 1.0000000000001e-40 |
| **power.c (X=10, Y = -350)**  
704 constraints | single: +0.0  
double: +0.0 | clpr: +0.0,  
clpq: +10^{-350}  
ic: [-4.94065645841247e-324, 4.94065645841247e-324] | single: +0.0  
double: +0.0 |
| **[Howden 82]**  
\(T_1 = A * B, X_1 = T_1 + 2, X_2 > 100, X_2 = 100\)  
\(-X_3, X_3 = X_3 - 50, X_3 > 50\)** | infeasible | clpr,clpq: infeasible  
ic: infeasible | infeasible |
Experimental results with FPSE

<table>
<thead>
<tr>
<th>#</th>
<th>NbC</th>
<th>NbV</th>
<th>Global results</th>
<th>On the solution path</th>
<th>ULP Max</th>
<th>Speedup factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NbE</td>
<td>NbD</td>
<td>NbV</td>
<td>NbE</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>12</td>
<td>62</td>
<td>17,515</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>22</td>
<td>3,948</td>
<td>484,128</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>32</td>
<td>461</td>
<td>102,522</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>59</td>
<td>42</td>
<td>544,377</td>
<td>9,208,097</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>73</td>
<td>52</td>
<td>510</td>
<td>158,716</td>
<td>52</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>87</td>
<td>62</td>
<td>799</td>
<td>209,621</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>101</td>
<td>72</td>
<td>494</td>
<td>87,934</td>
<td>72</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>115</td>
<td>82</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>9</td>
<td>129</td>
<td>92</td>
<td>258</td>
<td>83,166</td>
<td>92</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>143</td>
<td>102</td>
<td>637</td>
<td>157,421</td>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>157</td>
<td>112</td>
<td>224</td>
<td>73,702</td>
<td>112</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>171</td>
<td>122</td>
<td>635</td>
<td>153,318</td>
<td>122</td>
<td>0</td>
</tr>
</tbody>
</table>

The speedup due to ULP-Maximum does not depend on NbC or NbV!
Symbolic path-oriented test input generation on FP-computations is feasible!
Conclusions

- Testing for detecting rounding errors is important

- CP-based solvers for continuous domains can be tuned for FP constraints

- Our preliminary experiments with FPSE show that:
  1. ULP-Maximum is useful for solving FP constraints
  2. Symbolic path-oriented test input generation is feasible (up to 200 constraints on a path, in a couple of seconds)!

- But, more experiments to compare with SMT-solving are needed!
Thank you!