Symbolic Execution of Floating-Point Computations

A constraint-based testing approach

Bernard Botella
Thales Aerospace

Claude Michel
Projet COPRIN
INRIA Sophia-Antipolis

Arnaud Gotlieb
Projet LANDE - IRISA
INRIA Rennes

Part of the ACI V³F Project:
Validation & Véérification of floating-point computations

Partners:
- LIFC – INRIA Cassis (B. Legeard, …)
- I3S – INRIA Coprin (C. Michel, M. Rueher, …)
- IRISA – INRIA Vertecs / Lande (A. Gotlieb, T. Jéron, …)
- CEA – Lsl/List (B. Marre, M. Martel, E. Goubault, …)

http://lifc.univ-fcomte.fr/%7Ev3f/
Introduction

Symbolic Execution (SE): Evaluation of statements with symbolic values along a control flow path

SE is exploited in several applications:
- automatic test data generation [King 75, Clarke TSE’76, Meudec STVR’01]
- feasible path analysis [Goldberg et al. ISSTA’94]
- program proving [Chen et al. ISSTA’02]
- software model checking [Khushid et al. TACAS’03]
...

Currently, floating-point variables are handled as reals or rationals in SE applications.

```c
float foo( float x) {
    float y = 1.0e12, z;
    1. if( x < 10000.0 )
    2.  z = x + y;
    3. if( z > y)
    4.  ...

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

Path conditions:
- On the reals: $x \in (0,10000)$
- On the floats: no solution!

Path conditions:
- $x < 10000.0$
- $x + 1.0e12 > 1.0e12$
Conversely,

```c
float foo(float x) {
    float y = 1.0e12, z;
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: \( x \in (0, 32767.99\ldots) \)

Our roadmap

- To build a constraint solver over the floats
- To combine the solver over the floats with a finite domain constraint solver to deal with mixed computations
- To provide adequate labelling strategies over floating-point variables
Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
- Solving path conditions over the floats
- Current implementation FPSE
- Further work

Binary floating-point numbers (IEEE-754)

- float : (s,f,e) a bit pattern of 32, 64 or more bits
- 0 < e < e_{max} : Normalized
  \((-1)^s \cdot 1.f \cdot 2^{(e - \text{bias})}\)
  \begin{itemize}
    \item sign (1 bit)
    \item significand (23, 52 bits or extended)
    \item exponent (8, 11 bits or extended)
  \end{itemize}
- e = 0 : Denormalized \((-1)^s \cdot 0.f \cdot 2^{-(\text{bias} + 1)}\)
  \begin{itemize}
    \item +0.0, -0.0
  \end{itemize}
- e = e_{max} : +INF, -INF, NaNs

- 4 rounding modes (near, up, down, chop), 5 types of fp_exception
- Monotony of rounding required
**Accuracy requirement of IEEE-754**

For `add`, `sub`, `mult`, `div`, `sqrt`, `rem` and `conv`:
the floating-point result of an operation between floating-point numbers must be the rounding result of the exact operation over the reals.

Ex: `999999995904_f` `add` `10000_f` yields to `999999995904_f` = `near(999999995904_f + 10000_f)`

- Poor (but well-done) approximation of the reals
- finite set not uniformly distributed over the reals
- associativity and distributivity are lost in general,…

- Absorption `(x_f + ε = x_f)_f`, Cancellation `(X_f-Y_f = K.(X-Y) et K>>1)`

**Context of this work**

- Programs that strictly conform to IEEE-754

```
E ::= E add E | E subs E | E mult E | E div E
    | E == E | E != E | E > E | E >= E
    | (float) E | (double) E | Var | 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)
Outline

IEEE-754 and restrictions

Simple Symbolic Execution

Solving path conditions over the floats

Current implementation FPSE

Further work

Simple Symbolic Execution [Clarke 76]

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

\(\text{State} = \{<v, \phi>\}_{v \in \text{Var}(P)}\) \(\phi\) is an algebraic expr. over \(X\)

\(\text{PC} = c_1, \ldots, c_n\) a finite conjunction of conditions over \(X\) and temporary assignments
Symbolic state : features

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

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

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

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

(Path, State, PC) : examples

- \((1, \{<x, X>, <y, 1.0e12>, <z, \bot>\}, \text{true})\)
- \((1 \rightarrow 2 \rightarrow 3,\)  
  \(\{<x, X>, <y, 1.0e12>, <z, X + 1.0e12>\},\)  
  \(X < 10000.0\)\)
- \((1 \rightarrow 2 \rightarrow 3 \rightarrow 4,\)
  \(\{<x, X>, <y, 1.0e12>, <z, X + 1.0e12>\},\)  
  \(X < 10000.0, T := X + 1.0e12, T > 1.0e12)\)
Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
- Solving path conditions over the floats
- Current implementation FPSE
- Further work

Interval consistency over $F$ [C. Michel 02]

- $F_{\text{var}} x$ abstracted by an interval with FP bounds $I_x$
- Interval propagation:
  - Decomposing expressions into arity-3 constraints
    - $[v := x+y+z]$ leads to $\{t := x + y \mid v := t + z\}$
  - Conservative over-estimation of the solution-set — Formally speaking, FP-2B($w$)-consistency is achieved
Existing IP solvers over the reals

- BNR-Prolog [Older Vellino 93]
- Interlog [Botella Taillibert 93]
- K_b consistencies [Lhomme 93]
- Dynamic optimizations [Lhomme Gotlieb Rueher 96, 98]
- Newton [Van Hentenryck Michel Benhamou 94]
- Ilog Solver [Puget 94]
- Prolog IV [Colmeraurer 94]
- Numerica [Van Hentenryck 97]
- ECLIPSE(IC) [Wallace ... 97]
- Realpaver [Granvilliers 98]
- ...

Interval Arithmetic over the reals:

If \( I_x = [a,b] \) and \( I_y = [c,d] \) then

\[
I_{x+y} = [\text{down}(a+c), \text{up}(b+d)] \quad I_{x-y} = [\text{down}(a-d), \text{up}(b-c)]
\]

Our approach: floating-point projections

- **Direct and indirect projections for the assignment:**
  - \( \text{proj}(r, r := a + b) \) (direct)
  - \( \text{proj}(r, r := a + b) \) (1st indirect)
  - \( \text{proj}(r, r := a + b) \) (2nd indirect)

- **Direct projections** (over numeric fp numbers):
  - If \( I_r = [r_l, r_h] \), \( I_a = [a_l, a_h] \) and \( I_b = [b_l, b_h] \) then
  - \( r := a + b \) \( \rightarrow [r_l', r_h'] \leftarrow [a_l + b_l, a_h + b_h] \cap [r_l, r_h] \)
  - \( r := a - b \) \( \rightarrow [r_l', r_h'] \leftarrow [a_l - b_l, a_h - b_h] \cap [r_l, r_h] \)
  - \( r := a \times b \) \( \rightarrow [r_l', r_h'] \leftarrow [a_l a_h, b_l b_h] \cap [r_l, r_h] \)
  - \( r := a \div b \) \( \rightarrow [r_l', r_h'] \leftarrow [a_l / b_h, a_h / b_l] \cap [r_l, r_h] \)

... Recall that \( [x + y] \) denotes \( \text{near}(x + y) \)
**Ex: Direct projection**

\[
[r := a \text{ add } b]
\]

Monotony of rounding:

\(r_1 < r_2 \rightarrow \text{near}(r_1) < \text{near}(r_2)\)

---

**More complex : indirect projections**

If \(l_r = [r_l, r_u]\), \(l_a = [a_l, a_u]\) and \(l_b = [b_l, b_u]\) then

1st indirect projection of \([r := a \text{ add } b]\)

\[
[a'_l, a'_u] \leftarrow \text{mid}(r_l, r_l^-) \text{ subs } b_u, \text{ mid}(r_u, r_u^+ \text{ subs } b_l) \cap [a_l, a_u]
\]

1st indirect projection of \([r := a \text{ subs } b]\)

\[
[a'_l, a'_u] \leftarrow \text{mid}(r_l, r_l^-) \text{ add } b_l, \text{ mid}(r_u, r_u^+ \text{ add } b_l) \cap [a_l, a_u]
\]

2nd indirect projection of \([r := a \text{ subs } b]\)

\[
[b'_l, b'_u] \leftarrow [a_l \text{ subs } \text{mid}(r_u, r_u^+), a_u \text{ subs } \text{mid}(r_l, r_l^-)] \cap [b_l, b_u]
\]
Ex: 1st indirect projection $[r := a \text{ add } b]$

$$a_h' \leftarrow \min(mid(r_h, r_h^-) \text{ subs } b_h, a_h)$$

$$a_l' \leftarrow \max(mid(r_l, r_l^-) \text{ subs } b_h, a_l)$$

Impossible, as operations are correctly rounded

Impossible

Not optimal!

But computable with to-the-nearest

Handling comparisons and conversions

Comparisons (1st proj):

$$[a_l', a_h'] \leftarrow [\max(a_l, b_l), \min(a_h, b_h)] \quad \text{when } [a = b]$$

$$[a_l', a_h'] \leftarrow [\max(a_l, b_l^+), a_h] \quad \text{when } [a > b]$$

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

Floating-point conversions:

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

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

First isolate the singular bounds and then compute with an extended arithmetic defined by specific tables:

values of $a$ in 1st indirect projection of $[r := a + b]$

<table>
<thead>
<tr>
<th>$b$</th>
<th>$\text{-INF}$</th>
<th>$-0.0$</th>
<th>$+0.0$</th>
<th>$Nv$</th>
<th>$+\text{INF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{-INF}$</td>
<td>$Nv, -\text{INF} \pm 0.0$</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$-0.0$</td>
<td>$-\text{INF}$</td>
<td>$-0.0$</td>
<td>$+0.0$</td>
<td>$Nv$</td>
<td>$+\text{INF}$</td>
</tr>
<tr>
<td>$+0.0$</td>
<td>$-\text{INF}$</td>
<td>--</td>
<td>$\pm 0.0$</td>
<td>$Nv$</td>
<td>$+\text{INF}$</td>
</tr>
<tr>
<td>$Nv$</td>
<td>$Nv, -\text{INF}$</td>
<td>--</td>
<td>$Nv, \pm 0.0$</td>
<td>$Nv, \pm 0.0$</td>
<td>$Nv, +\text{INF}$</td>
</tr>
<tr>
<td>$+\text{INF}$</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>$Nv, +\text{INF}, \pm 0.0$</td>
</tr>
</tbody>
</table>

NaNs (for example $\text{INF} - \text{INF}$) are currently not handled (Prolog’s fail as result)

Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
- Solving path conditions over the floats
  - Current implementation: FPSE
- Further work
FPSE: Floating-Point Symbolic Execution

- Implemented as part of the INKA tool
  - B. Botella: full design and first implementation
  - Me: constraint propagation engine and improvements on fp projection functions

- Handles C computations for gcc/solaris/sparc and visual/xp/intel (unsound w.r.t. the stack of 80-bits registers)

- Written in SICStus Prolog (constraint propagation engine) and C (floating-point projection functions)

- Integrated with a collaborative integer constraint solver (SICStus clpfd library)

A straightforward collaboration principle

- Synchronous trigger mechanism between FPSE and clpfd
- Communicates through alarms (fd_global mechanism) when int-to-float or float-to-int conversion constraints are encountered

\[ I \leftarrow (\text{long}) \times \]
\[ \text{tell}(I \in \text{min}_{FD}(X)\ldots\text{max}_{FD}(X)) \]
\[ \text{Alarm}(\text{dom}(J) \text{ is pruned}) \]
\[ Y \leftarrow (\text{float}) \times J \]
\[ \text{tell}(Y \in \text{min}_{FP}(J)\ldots\text{max}_{FP}(J)) \]
An example (extracted from www.astree.ens.fr/)

```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);
}
```

```prolog
test24 :-
  solveur:init_env(E),
  flottant:news([Y,Z,R],float(32),['y','z','r'],E),
  flottant:news([X,C,T1,T2],double(64),['x','c','t1','t2'],E),
  flottant:affect(const('1125899973951488.0'),X),
  flottant:affect(const('1.0'),C),
  flottant:affect('+',X,C,T1),
  flottant:affect(conv(double(64),float(32)),T1,Y),
  flottant:affect('-',X,C,T2),
  flottant:affect(conv(double(64),float(32)),T2,Z),
  flottant:affect('-',Y,Z,R),
  solveur:solve(E),
  flottant:fprint([R]).
\%
| ?- test24.
\%
```

Selected experimental results (gcc/solaris/sparc)

<table>
<thead>
<tr>
<th>Programs</th>
<th>Expected results</th>
<th>Eclipse</th>
<th>FPSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Goldberg 91] 2.0e-30 + 1.0e30 - 1.0e-30</td>
<td>single: -1.000000003e-30 double: -1.0e-30</td>
<td>single: -1.000000003e-30 double: -1.0e-30</td>
<td></td>
</tr>
<tr>
<td>X &lt; 1.0e4, T1=X + 1.0e12, T2=1.0e12</td>
<td>single: infeasible path double: [0.103e-6, 9.999e-3]</td>
<td>single: infeasible path double: [0.103e-6, 9.999e-3]</td>
<td></td>
</tr>
<tr>
<td>X &gt; 0, Y = X + 1.0e12, T1=1.0e12</td>
<td>single: infeasible path double: [0.103e-6, 9.999e-3]</td>
<td>single: infeasible path double: [0.103e-6, 9.999e-3]</td>
<td></td>
</tr>
<tr>
<td>power.c (X&gt;10, Y = -40) 84 constraints</td>
<td>single: +0.0 double: 1.0000000000e-40</td>
<td>single: +0.0 double: 1.0000000000e-40</td>
<td></td>
</tr>
<tr>
<td>power.c (X&gt;10, Y = -350) 794 constraints</td>
<td>single: +0.0 double: +0.0</td>
<td>single: +0.0 double: +0.0</td>
<td></td>
</tr>
<tr>
<td>[Howden 82] 10 &lt; X &lt; 100, X^2 + 2.X &gt; 100, X &gt; 50, X = 50.</td>
<td>single: infeasible double: infeasible</td>
<td>single: infeasible double: infeasible</td>
<td></td>
</tr>
</tbody>
</table>
Problems: weakness of fp\_constraints to prune the domains

example adapted from \[KPV\ TACAS'03\] :

```prolog
int f(float x, float y) {
    if (x > y) {
        x = x+y;
        y = x-y;
        x = x-y;
    }
    if (x = y > 0) ...
```

Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
- Solving path conditions over the floats
- Current implementation: FPSE
- Further work
Summary and further work

Contribution: New results to solve accurately path conditions over floating-point computations
- Symbolic values
- Comparisons and conversions

An implementation: FPSE

Application: Structural test data generation

Extensions: NaNs
- Transcendental functions (\(\text{sqrt, sin, exp, \ldots}\))

Problems: Improving the pruning capacity of projection functions
- Labelling process (heuristics)

The Floats...
...source of inspiration?

1. B. Botella, A. Gottlieb, C. Michel
   «Symbolic execution of floating-point computations»

2. B. Botella, A. Gottlieb
   «Documentation of a floating-point interval constraint solver » -- V3F Internal report
Playing with FPLIB: a first experience

- **Objectives:**
  - β-testing of FPLIB
  - comparing the results of FPLIB with those of FPSE (when appropriate!)

- **Results:**
  - better understanding of implementation choices
  - five bugs detected in FPLIB, one bug in FPSE
  - five deliveries of FPLIB: Thanks to Claude!

But, experimental comparison still in progress

---

<table>
<thead>
<tr>
<th></th>
<th>FPSE</th>
<th>FPLIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Bernard Botella (Thales) and Arnaud Gottlieb (IRISA)</td>
<td>Claude Michel (Coprin)</td>
</tr>
<tr>
<td>Language</td>
<td>Sicstus Prolog, C</td>
<td>2 (static) C++ libraries</td>
</tr>
<tr>
<td>Target</td>
<td>GCC / Solaris / Sparc (+ Visual / XP / Intel)</td>
<td>GCC / Linux / Intel</td>
</tr>
<tr>
<td>Formats</td>
<td>single, double</td>
<td>single, double, long double</td>
</tr>
<tr>
<td>Rounding</td>
<td>to the nearest</td>
<td>to zero, to +∞, to -∞, to the nearest</td>
</tr>
<tr>
<td>Operators</td>
<td>+, -, *, /, ==, !=, &lt;=, &lt;=, int-float + float-double conv</td>
<td>+, -, *, /, ==, !=, &lt;=, &lt;=, float-double-extended conv</td>
</tr>
<tr>
<td>Functions</td>
<td>no</td>
<td>sqrt, fabs, exp, log, sin, cos, tan, asin, acos, atan, cbrt, ..</td>
</tr>
<tr>
<td>Automatic decomposition</td>
<td>no</td>
<td>yes – C syntax</td>
</tr>
<tr>
<td>Collaboration with other solvers</td>
<td>Sicstus clp(fd)</td>
<td>no</td>
</tr>
</tbody>
</table>
FPSE and FPLIB with automatic decomposition are incomparable

\[ \text{exp} := 1.0e12f + 32000.0f + 32000.0f \quad \text{with to-the-nearest rounding mode} \]

In FPSE:

```prolog
test27 :-
    solveur:init_env(E),
    flottant:news([A,B,C,EXP,TMP], float(32), ['a','b','c','exp','tmp'], E),
    flottant:affect(const('1.0e12'), A),
    flottant:affect(const('32000.0'), B),
    flottant:affect(const('32000.0'), C),
    flottant:affect('+', A, B, TMP),
    flottant:affect('+', TMP, C, EXP),
    solveur:solve(E),
    flottant:affiche(EXP).
?- test27. /* on sun4u sparc SUNW, Ultra-250 */
```

\[ \text{float}(32) : \text{exp} \in 9.999999959040000000e+11 .. 9.999999959040000000e+11 \]

In FPLIB:

```c
int main(void) {
    int nbsols = 0;   Fpc_Csp CSP;
    Fpc_Variable a(CSP, "a", FPC_FLOAT);
    ...
    Fpc_Model model(CSP);
    model.add(a = 1.0e12);
    model.add(b = 32000.0);
    model.add(c = 32000.0);
    model.add(exp = a + b + c);
    model.extract();
    Fpc_Solver solver(model);
    if( ! solver.TwoB(0.0) ) printf("pas de solution !\n");
    for (int i = 0; i < CSP.variables.size(); i++) {
        CSP.variables[i]->display(); printf("\n");
    }
}
```

\[ \text{exp} \in [ 1.000000061440000000e+12, 1.000000061440000000e+12]f \]

---

direct projection of a basic operation

\[ \text{[r := a add b]} \]

FPSE models only the to-the-nearest mode

FPLIB with SetRoundDown()
inverse projection of a basic operation

\[ r := a \text{ add } b \]

\[ a' \leftarrow \max( \text{mid}(r_l, r_{l+}), \text{subs } b_h, a) \]

where:

\[ \theta^+(x) = x^+ \text{ if } x \text{ is a float} \]

\[ = 0^- \times x \text{ otherwise} \]
### Summary

**FPLIB:**

- transcendental
- automatic decomposition
- rounding mode / precision
- documentation
- source code?

### Very first experimental comparison

<table>
<thead>
<tr>
<th>Programs</th>
<th>Expected results</th>
<th>FPLIB</th>
<th>FPSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = \log X, Y = \sin X$</td>
<td>[1.4012984643248117e-45, 3.2767998046875000e+04]</td>
<td>single: infeasible path</td>
<td>single: infeasible path</td>
</tr>
<tr>
<td>$\text{area} = 1.9337656497955322$</td>
<td>[4.9406564584124654e-324, 6.1035156250000000e-05]</td>
<td>double: $[6.103\pm5, 9.999\pm3]$</td>
<td>double: $[6.103\pm5, 9.999\pm3]$</td>
</tr>
<tr>
<td>$\text{trap computes the area of sin between 0 and } \pi$</td>
<td>[4 solutions]</td>
<td>single: infeasible path</td>
<td>double: $[6.103\pm5, 9.999\pm3]$</td>
</tr>
</tbody>
</table>

*Note: Single precision*
**Example:** \( y = \log(x) \), \( x+y = 0 \)

4 projection functions

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

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

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

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

Conservative over-estimation of the solution-set – Formally speaking, \( 2-b(w) \)-consistency is achieved

**Notations**

- \( x \) near(\( x \))
- \( a^+ \)
- \( a^- \)
- \( \text{mid}(a, a^+) \)