Correct Handling of Floating-Point Computations in Symbolic Execution

Bernard Botella
THALES A.S.

Arnaud Gotlieb
IRISA / INRIA

Claude Michel
INRIA Sophia-Antipolis
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]
- model checking [Khushid et al. TACAS’03]
...

Currently, floating-point variables are handled as reals in SE applications
float foo(float x) {
    float y = 1.0e12;
    1. if (x < 10000.0) 
    2. z = x + y;
    3. if (z > y)
    4. ...

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

Path conditions:

x < 10000.0
x + 1.0e12 > 1.0e12

On the reals: x ∈ (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: \(x \in (0, 32767.99\ldots)\)
Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
- Solving path conditions over the floats
- Current implementation FPSE
- Further work
Floating-point numbers (IEEE-754)

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

- \(0 < e < e_{\text{max}}\): Normalized
  
  \((-1)^s \ 1.f \ 2^{(e - \text{bias})}\)

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

- \(e = e_{\text{max}}\): +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 \texttt{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: \(999999995904_f \text{ add } 10000_f \) yields to \(999999995904_f\)

\[= \text{near}(999999995904_f + 10000_f)\]

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

- Absorption \((X_f + \varepsilon == X_f)\), Cancellation \((X_f-Y_f==K.(X-Y) \text{ et } K>>1)\)
Context of this work

Programs 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 near-to-even 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

Notations: Control Flow Graph \((N, A, e, s)\)

- \(x\) vector of symbolic input

Definition (Symbolic State):

\((\text{Path}, \text{State}, \text{PC})\) where

- \(\text{Path} = n_1 \ldots n_j\) 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, \perp>\}, 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 + 1.0e12, T > 1.0e12
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 \(S_{PC}\) be the solution-set of \(PC\)
  then \(\forall X \in S_{PC}, \text{ Path is sensitized by } X\)

- When \(S_{PC}=\emptyset\) then \(\text{Path is non-feasible}\) undecideable in the general case \([\text{Weyuker 79}]\)
Outline

- IEEE-754 and restrictions
- Simple Symbolic Execution
  - Solving path conditions over the floats
- Current implementation FPSE
- Further work
Non-linear constraint solving over $\mathbb{R}$ with interval propagation [Cleary 87, Davis 87]

- Real var $x$ abstracted by an interval with FP bounds $I_x$
- Interval Arithmetic over the reals:
  $I_x = [a,b]$ and $I_y = [c,d]$ then
  $I_{x+y} = [\text{down}(a+c), \text{up}(b+d)]$
  $I_{x-y} = [\text{down}(a-d), \text{up}(b-c)]$
  $I_{\exp(x)} = [\text{down}(\exp(a)), \text{up}(\exp(b))]$

- Relations over intervals / decomposition into proj. fcts:

\[
\begin{align*}
  I_z' &\leftarrow I_{x+y} \cap I_z \\
  I_x' &\leftarrow I_{z-y} \cap I_x \\
  I_y' &\leftarrow I_{z-x} \cap I_y
\end{align*}
\]

- Constraint propagation
Example: \( y = \log(x), \ x+y = 0 \)

4 projection functions

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

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

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

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

Conservative over-estimation of the solution-set – Formally speaking, \( 2-b(w)\)-consistency is achieved
A few existing solvers based on IP

- BNR-Prolog  [Older & Vellino 93]
- Interlog  [Botella & Taillibert 93]
  \[ \text{K}_b\_\text{consistencies} \quad \text{[Lhomme 93]} \]
  \[ \text{Dynamic optimizations} \quad \text{[Lhomme Gotlieb Rueher 96, 98]} \]
- Newton  [Benhamou et al. 94]
- Ilog Solver  [Puget 94]
- Prolog IV  [Colmerauer 94]
- Numerica  [Van Hentenryck 97]

...
Our approach to solve path conditions: Interval propagation over floating-point variables

- Notations:
  - Recall that $[a + b]$ denotes $[\text{near}(a + b)]$
  - Path conditions are made of constraints and assignments

```
Recall that [a add b] denotes [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 \( I_r = [r_l, r_h], \ I_a = [a_l, a_h] \) and \( I_b = [b_l, b_h] \) 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

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

\[
[a_l', a_h'] \leftarrow [\text{mid}(r_l, r_l^-) \, \text{subs} \, b_h, \text{mid}(r_h, r_h^+) \, \text{subs} \, b_l] \cap [a_l, a_h]
\]

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

\[
[a_l', a_h'] \leftarrow [\text{mid}(r_l, r_l^-) \, \text{add} \, b_l, \text{mid}(r_h, r_h^+) \, \text{add} \, b_h] \cap [a_l, a_h]
\]

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

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

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

impossible

not optimal, but computable with near-to-even
Handling comparisons and conversions

Comparisons (1st proj):

\[
[a_1', a_h'] \leftarrow [\max(a_l, b_l), \min(a_h, b_h)]
\]
when \([a==b]\)

\[
[a_1', a_h'] \leftarrow [\max(a_l, b_l)^+, a_h]
\]
when \([a > b]\)

\[
[a_1', a_h'] \leftarrow [\text{if}(a_l=b_l=b_h) \text{ then } a_l^+ \text{ else } a_l, \\
\text{if}(a_h=b_l=b_h) \text{ then } a_h^- \text{ else } a_h]
\]
when \([a! = b]\)

Floating-point conversions:

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

\[
[r_1', r_h'] \leftarrow [\max_f((\text{float})a_l, r_l), \min_f((\text{float})a_h, r_h)]
\]
(direct proj.)

\[
[a_1', a_h'] \leftarrow [\max_d(a_l, \text{mid}(r_l, r_l^-)), \min_d(a_h, \text{mid}(r_h, r_h^-))]
\]
(indirect)
Handling zeros and infinities

Based on an extended arithmetic defined by specific tables:

<table>
<thead>
<tr>
<th>b \ r</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,</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>-\text{INF},\pm0.0</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>\pm0.0</td>
<td>Nv</td>
<td>+\text{INF}</td>
</tr>
<tr>
<td>Nv</td>
<td>Nv, -\text{INF}</td>
<td>--</td>
<td>Nv,\pm0.0</td>
<td>Nv,\pm0.0</td>
<td>Nv, +\text{INF}</td>
</tr>
<tr>
<td>+\text{INF}</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Nv, +\text{INF},\pm0.0</td>
</tr>
</tbody>
</table>
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 as the INKA project (by Thales A.S.)
  Bernard : full design and implementation
  Me : a few improvements on fp projection functions

- Handles C computations on Sparc and Intel

- Written in 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 conversion constraints are encountered

FPSE

clpfd

I ← (long) X
X ← (float) J

tell( I in min(X)..max(X))
alarm(dom(J) is pruned)
```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
```

```
/* 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
```

```
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(in(1125899973951488.0, 1125899973951488.0),X),
  flottant:affect(in(1.0, 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]).
```

An example (extracted from www.astree.ens.fr/)
### Very, very first experimental results (extracts)

<table>
<thead>
<tr>
<th>Programs</th>
<th>Path conditions</th>
<th>in</th>
<th>clpr</th>
<th>clpq</th>
<th>FPSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Goldberg 91]</td>
<td>$(2. \cdot 10^{-30} + 1 \cdot 10^{-30}) - 1. \cdot 10^{-30}$</td>
<td>X</td>
<td>0.0</td>
<td>1/9999..</td>
<td>$-1.00000000$</td>
</tr>
<tr>
<td></td>
<td>$T_2 := 2 \cdot 10^{-30} \text{ add } 1 \cdot 10^{-30},$</td>
<td></td>
<td></td>
<td></td>
<td>$317107e-30$</td>
</tr>
<tr>
<td></td>
<td>$T_1 := T_2 \text{ subs } 1. \cdot 10^{-30},$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T_3 := T_1 \text{ subs } 1. \cdot 10^{-30}, X = T_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Goldberg 91]</td>
<td>$D := B^2 - 4AC$</td>
<td>D</td>
<td>0.029200..</td>
<td>73/2500</td>
<td>0.029199600</td>
</tr>
<tr>
<td></td>
<td>$A := 1.22, B := 3.34, C := 2.28$</td>
<td></td>
<td></td>
<td></td>
<td>219726562</td>
</tr>
<tr>
<td></td>
<td>$T_1 := B \text{ mult } B, T_2 := A \text{ mult } C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T_3 := 4. \cdot 4 \text{ mult } T_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D := T_1 \text{ subs } T_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Goldberg 91]</td>
<td>$D := B^2 - 4AC$</td>
<td>C</td>
<td>2.285983.7</td>
<td>27889/12200</td>
<td>Interval of 8 single floats + 2.285982..4</td>
</tr>
<tr>
<td></td>
<td>$A := 1.22, B := 3.34, D := +0.$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T_1 := B \text{ mult } B, T_2 := A \text{ mult } C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$T_3 := 4. \cdot 4 \text{ mult } T_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D := T_1 \text{ subs } T_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X &lt; 10000., T_1 := X \text{ add } 1. \cdot 10^{12}$, $T_1 &gt; 1. \cdot 10^{12}$</td>
<td>X</td>
<td>(-0.,10000)</td>
<td>(0,10000)</td>
<td>infeasible</td>
<td></td>
</tr>
<tr>
<td>$X &gt; 0.0, T_1 := X \text{ add } 1. \cdot 10^{12}$, $T_1 = 1. \cdot 10^{12}$</td>
<td>X</td>
<td>infeasible</td>
<td>infeasible</td>
<td>[1.4012..e45, 32768.0]</td>
<td></td>
</tr>
</tbody>
</table>

**Programs intended to run on Pentium 4 over the singles, compiled by Visual C++ with /Op**
Problems (1)

- FPSE must be runned on the same machine than the tested program to get the same rounding algorithms

- NaNs (for example INF – INF) are currently not handled, Prolog’s fail as result

- In practice, it is required to check a given result as:
  lack of documentation for the compiler design,
  hardware optimizations (such as the fused mult-add a*b+c),
  possible change of rounding modes, ..
Problems (2)
Weaknesses 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)

```

```
test12(T) :-
    solveur:init_env(E),
    flottant:news([X,Y,X1,X2,Y1,T1,C1],T,_,E),
    entier:new(R,int(32),'r',E),
    flottant:affect('>',X,Y,R),
    flottant:affect('+',X,Y,X1),
    flottant:affect('-',X1,Y,Y1),
    flottant:affect('-',X2,Y1,T1),
    flottant:affect(const('0.0'),C1),
    flottant:affect('>',T1,C1,R), R = 1,
    flottant:affect(in(+1.0E+30, +1.001E+30),X),
    flottant:affect(in(+1.0E+30, +1.001E+30),Y),
    solveur:solve(E),
    flottant:fprint([X,Y]).
```

The results provided by FPSE~:
float(32):x in 1.00000009060533E+30..1.001E+30
float(32):y in 1.0E+30..1.000999947816015E+30
Contributions and further work

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

Implementation is in progress: FPSE

Application: Structural test data generation

Extensions: NaNs

Transcendental functions (sqrt, sin, exp, ...)

Problems: Improving the pruning capacity of projection functions
- Labelling process (heuristics)
- Experimental evaluation
The Floats...

...source of inspiration?

1. B. Botella, A. Gotlieb, C. Michel
   «Correct handling of floating-point computations in symbolic execution»
   RR-INRIA 5150, mar. 2004, 24pp

2. B. Botella, A. Gotlieb
   «Documentation of a floating-point interval constraint solver» -- V3F Internal report