ICST 2013, Luxembourg, Apr. 2013
Dagsthul 2014: « Symbolic methods and constraint solving »

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

Arnaud Gotlieb

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

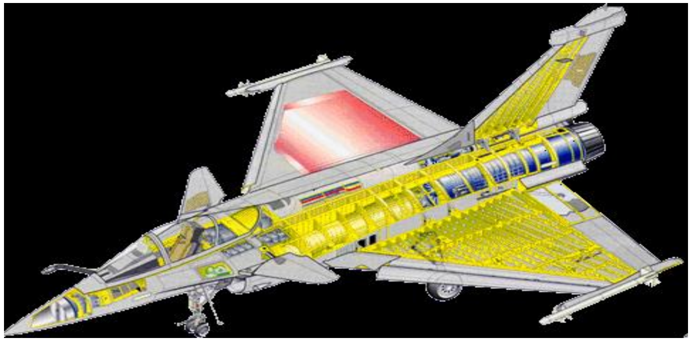
Joint work with Roberto Bagnara (Parma), Matthieu Carlier (IRISA) and Roberta Gori (Pisa)

Certus

sfi Centre for Research-based Innovation

Motivations

- ❑ Increasing use of ***floating-point computations*** in safety-critical systems



BCE Rafale – Dassault

Nuclear Power Plant - EDF

Alarm system - KM

- ❑ 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 → path conditions → constraint solving → test input

- However, handling ***correctly*** floating-point computations in constraint solving is difficult

```
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 ?

<u>Path conditions:</u>

$x < 10000.0$

$x + 1.0e12 > 1.0e12$

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

On the floats: no solution!

Conversely,

```
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 ?

<u>Path conditions:</u>

$x > 0.0$

$x + 1.0e12 = 1.0e12$

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

  $\underline{0 < e < e_{max}}$: Normalized

  significand (23, 52 bits or extended)

  $$(-1)^s \ 1.f \ \ 2^{(e - \mathit{bias})}$$

  sign (1 bit)

  exponent (8, 11 bits or extended)

  $\underline{e = 0}$: Denormalized $(-1)^s \ 0.f \ \ 2^{(-\mathit{bias} + 1)}$

  +0.0, -0.0

  $\underline{e = e_{max}}$: +INF, -INF, NaNs

- Rounding: **r('1.0e12') = $\mathbf{999999995904.0_f}$**

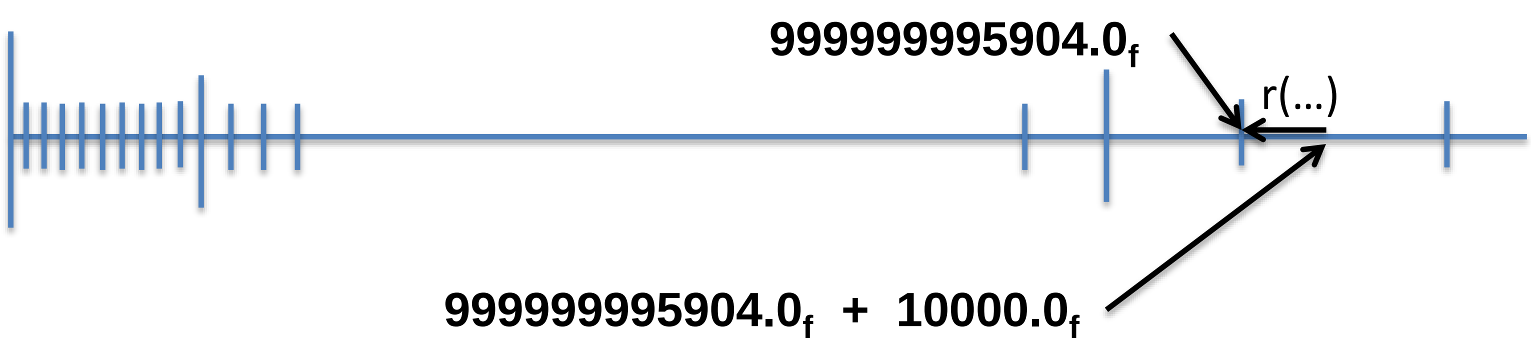
  4 modes (near-to-even, …), monotonicity (i.e., if x > y then r(x) > r(y))

Accuracy requirement of IEEE-754

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

$$
\begin{aligned}
\text{`1.0e12'} \;\; \text{add} \;\; \text{`10000.0'} &= r(\text{`1.0e12'}) \;\; \text{add} \;\; r(\text{`10000.0'}) \\
&= 999999995904.0_f \;\; \text{add} \;\; 10000.0_f \\
&= r(999999995904.0_f + 10000.0_f) \\
&= r(\text{`1000000005904.0'}) \\
&= 999999995904.0_f \\
&= \text{`1.0e12'}
\end{aligned}
$$

Poor (but well-conceived) approximation of the reals



Decomposition in symbolic execution

- Decomposition in SSA-like three-address code, preserving evaluation order

  e.g., z := z * z + z → t1 **==** z1 **mul** z1, z2 **==** t1 **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

$$
\begin{aligned}
E \quad ::= \; & E \texttt{ add } E \mid E \texttt{ subs } E \mid E \texttt{ mult } E \mid E \texttt{ div } E \\
& \mid E \texttt{ == } E \mid E \texttt{ != } E \mid E \texttt{ > } E \mid E \texttt{ >= } E \\
& \mid \texttt{(float) } E \mid \texttt{(double) } E \mid \mathit{Var} \mid \mathit{Constants}
\end{aligned}
$$

- 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 [Clarke 76]

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

X vector of symbolic input

Definition (*Symbolic State*) :

$(Path, State, PC)$ where

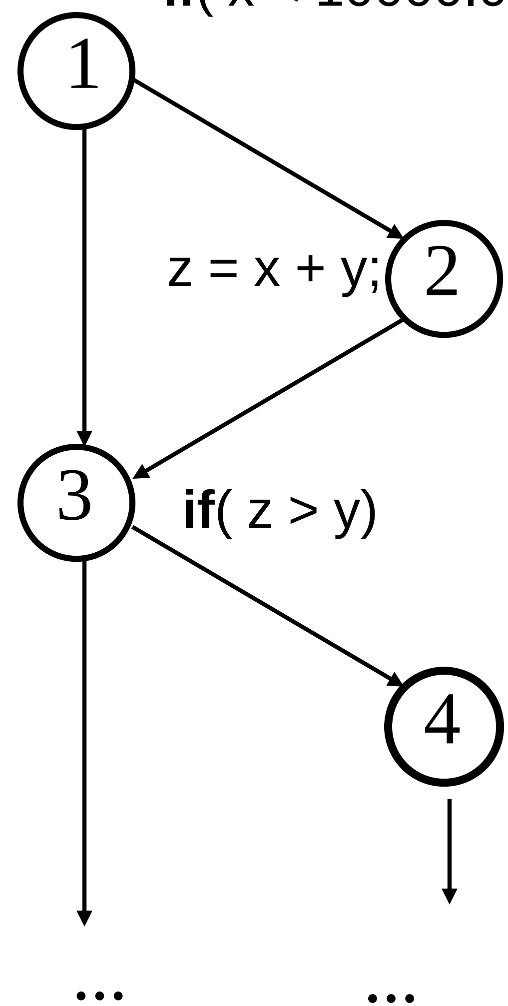
$Path = n_i \rightarrow .. \rightarrow n_j$ is a (partial) path of the CFG

$State = \{\langle v, \varphi \rangle\}_{v \in Var(P)}$ φ is an algebraic expr. over X

$PC = c_1, \ldots, c_n$ a finite conjunction of conditions over X or a temporary assignments

`(Path,` **`State`**`,PC)` : examples

float foo(**float** x) {
float y = 1.0e12 ;
if(x < 10000.0)



(1,**{<x,X>,<y,1.0e12>,<z,⊥>},**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)

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`
  then $\forall X \in S_{PC}$, `Path` is activated by `X`

- When $S_{PC} = \emptyset$ then `Path` is non-feasible

  However, finding all the non-feasible paths is a classical undecideable 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

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

Filtering, constraint propagation and labelling ➔ constraint solving

Example : $y = \log(x), x+y = 0$

4 projection functions

$$\left\{\begin{array}{llll} I_{x'} \leftarrow I_{\exp(y)} & \cap & I_x & ① \\ I_{y'} \leftarrow I_{\log(x)} & \cap & I_y & ② \\ I_{x'} \leftarrow I_{-y} & \cap & I_x & ③ \\ I_{y'} \leftarrow I_{-x} & \cap & I_y & ④ \end{array}\right.$$

$$\begin{array}{lccccc} & \xrightarrow{①,④} & & \xrightarrow{①,④} & & \xrightarrow{①,④} & & \xrightarrow{①,④} \\ x \in & [-\infty,+\infty] & & [0,+\infty] & & [0,\ 1] & & [0.56,\ 1] & & [0.56,\ 0.57] \\ y \in & [-\infty,+\infty] & & [-\infty,\ 0] & & [-1,\ 0] & & [-1,\ -0.56] & & [-0.57,\ -0.56] \end{array}$$

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)

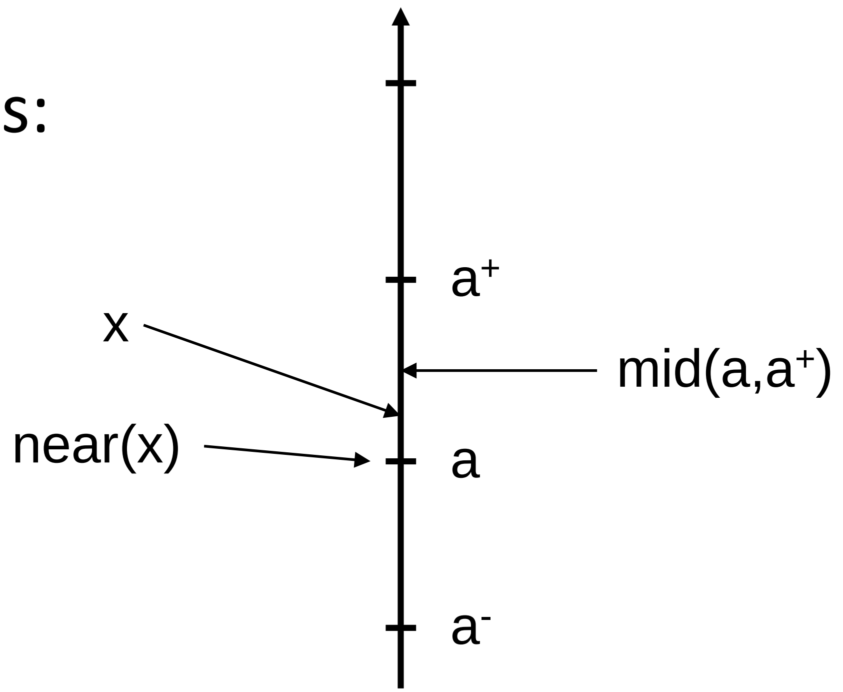
- NUMERICA (Van Hentenryck 1997)
- REALPAVER (Granvilliers 1998)

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 `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:

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

- proj(r, r:= a add b) (direct)
- proj(a, r:= a add b) (1^{st} indirect)
- proj(b,r := a add b) (2^{nd} indirect)

⌘ **Direct projections** (over numeric fp numbers):

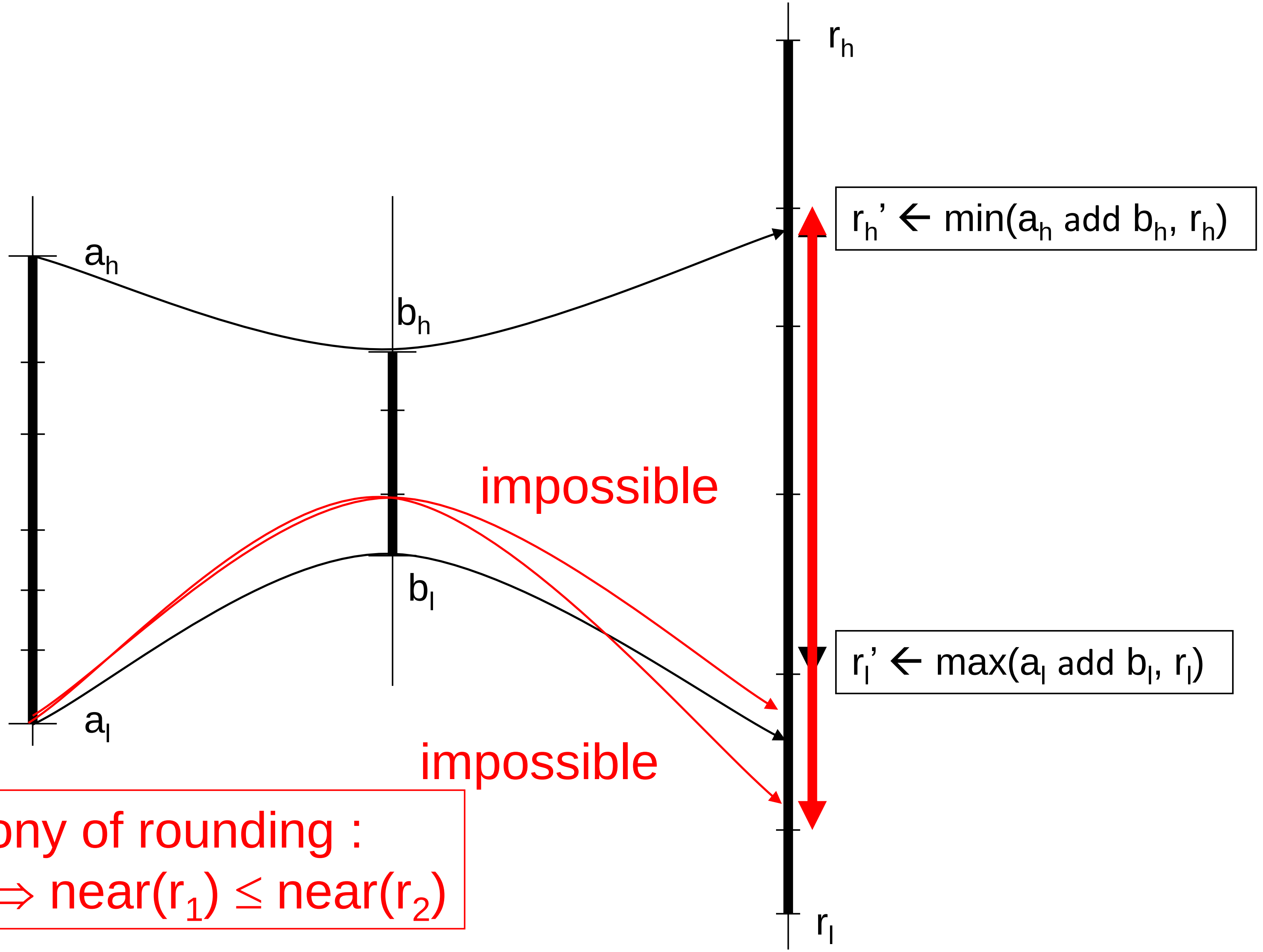
If $I_r = [r_l,r_h]$, $I_a = [a_l,a_h]$ and $I_b = [b_l,b_h]$ then

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

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

...

Ex: Direct projection `[r := a 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

<u>1st indirect projection of [r := a add b]</u>

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

<u>1st indirect projection of [r := a subs b]</u>

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

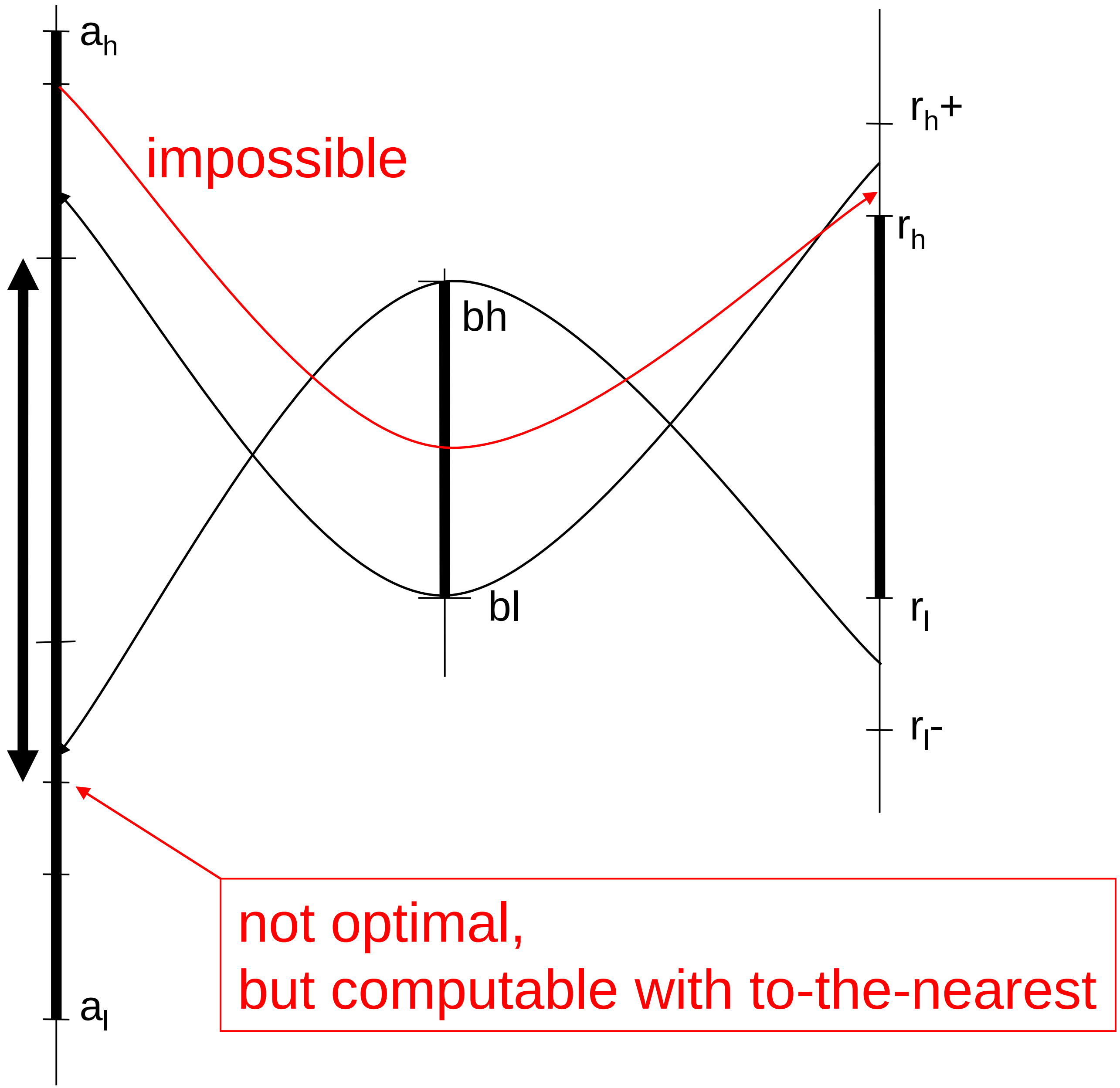
<u>2nd indirect projection of [r := a subs b]</u>

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

Ex: 1st indirect projection `[r := a add b]`

$a_h' \leftarrow \min($
$\mathrm{mid}(r_h, r_h^+)$ subs b_l,
$a_h)$

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



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 := (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.)

$[a_l', a_h'] \leftarrow [\max_d(al, \text{mid}(r_l, r_l^-)), \min_d(a_h, \text{mid}(r_h, r_h^+))]$ (indirect)

Handling zeros and infinities

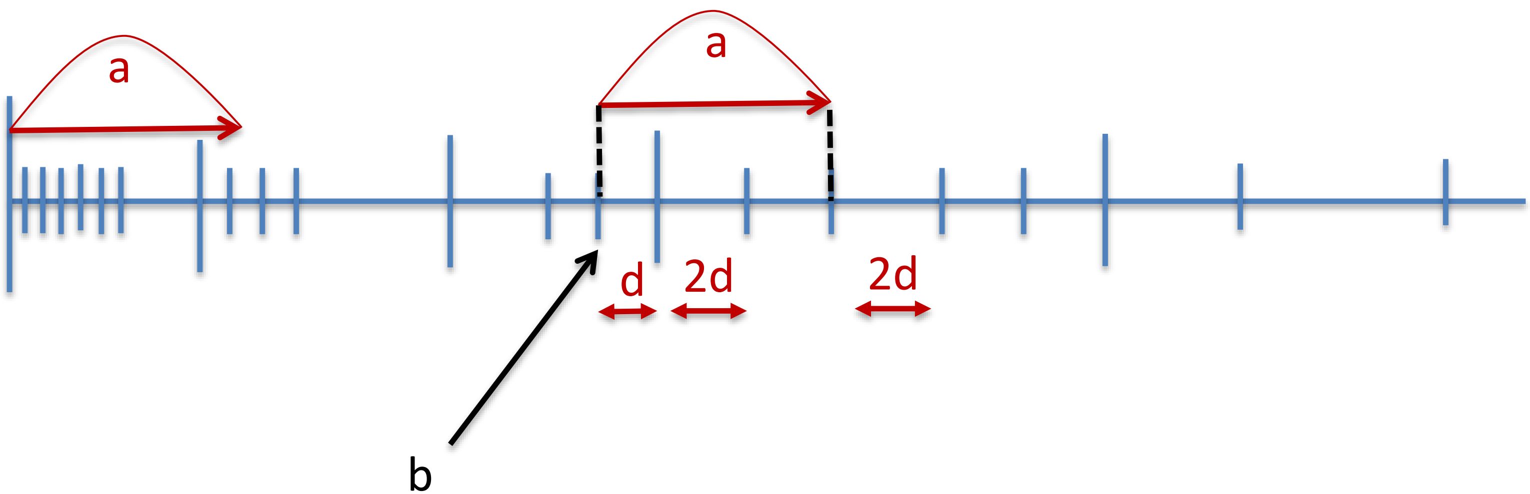
Based on an extended arithmetic defined by specific tables:

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

$_b$ \ ^{r}	-INF	-0.0	+0.0	Nv	+INF
-INF	Nv, -INF,±0.0	--	--	--	--
-0.0	-INF	-0.0	+0.0	Nv	+INF
+0.0	-INF	--	±0.0	Nv	+INF
Nv	Nv,-INF	--	Nv,±0.0	Nv,±0.0	Nv,+INF
+INF	--	--	--	--	Nv,+INF,± 0.0

The Marre&Michel property **(Marre and Michel 2010)**

$$\boxed{X - Y = a}$$



Then, The property says that Y cannot be greater than b

1. We have reformulated and corrected this property ➔ ULP-Maximum

Filtering by ULP-Maximum

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

Constraint	$\mathrm{x} \subseteq \cdot$	$\mathrm{y} \subseteq \cdot$	Condition(s)
$\mathrm{z} = \mathrm{x} \oplus \mathrm{y},\ 0 < \mathrm{z} \leq f_{\max}$	$[\ \underline{\delta}_{\oplus}(\zeta),\ \ \overline{\delta}_{\oplus}(\zeta)]$	$[\ \underline{\delta}_{\oplus}(\zeta),\ \ \overline{\delta}_{\oplus}(\zeta)]$	$\zeta = \mu_{\oplus}(\mathrm{z}),\quad -f_{\max} \leq \underline{\delta}_{\oplus}(\zeta),\quad \overline{\delta}_{\oplus}(\zeta) \leq f_{\max}$
$\mathrm{z} = \mathrm{x} \oplus \mathrm{y},\ -f_{\max} \leq \mathrm{z} < 0$	$[-\overline{\delta}_{\oplus}(\zeta'), -\underline{\delta}_{\oplus}(\zeta')]$	$[-\overline{\delta}_{\oplus}(\zeta'), -\underline{\delta}_{\oplus}(\zeta')]$	$\zeta' = \mu_{\oplus}(-\mathrm{z}),\quad -f_{\max} \leq \underline{\delta}_{\oplus}(\zeta'),\quad \overline{\delta}_{\oplus}(\zeta') \leq f_{\max}$
$\mathrm{z} = \mathrm{x} \ominus \mathrm{y},\ 0 < \mathrm{z} \leq f_{\max}$	$[\ \underline{\delta}_{\oplus}(\zeta),\ \ \overline{\delta}_{\oplus}(\zeta)]$	$[-\overline{\delta}_{\oplus}(\zeta), -\underline{\delta}_{\oplus}(\zeta)]$	$\zeta = \mu_{\oplus}(\mathrm{z}),\quad -f_{\max} \leq \underline{\delta}_{\oplus}(\zeta),\quad \overline{\delta}_{\oplus}(\zeta) \leq f_{\max}$
$\mathrm{z} = \mathrm{x} \ominus \mathrm{y},\ -f_{\max} \leq \mathrm{z} < 0$	$[-\overline{\delta}_{\oplus}(\zeta'), -\underline{\delta}_{\oplus}(\zeta')]$	$[\ \underline{\delta}_{\oplus}(\zeta'),\ \ \overline{\delta}_{\oplus}(\zeta')]$	$\zeta' = \mu_{\oplus}(-\mathrm{z}),\quad -f_{\max} \leq \underline{\delta}_{\oplus}(\zeta'),\quad \overline{\delta}_{\oplus}(\zeta') \leq f_{\max}$
$\mathrm{z} = \mathrm{x} \otimes \mathrm{y},\ 0 < \lvert z \rvert \leq 2(2 - 2^{1-p})$	$[\ \underline{\delta}_{\otimes}(m),\ \ \overline{\delta}_{\otimes}(m)]$	$[\ \underline{\delta}_{\otimes}(m),\ \ \overline{\delta}_{\otimes}(m)]$	$m = \max\{\lvert\underline{\mathrm{z}}\rvert, \lvert\overline{\mathrm{z}}\rvert\},$
$\mathrm{z} = \mathrm{x} \oslash \mathrm{y},\ 0 < \lvert\mathrm{z}\rvert \leq 1$	$[\ \underline{\delta}_{\oslash}(m),\ \ \overline{\delta}_{\oslash}(m)]$		$m = \max\{\lvert\underline{\mathrm{z}}\rvert, \lvert\overline{\mathrm{z}}\rvert\}$

$$\overline{\delta}_{\oplus}(z) = \begin{cases} \beta, & \text{if } 0 < z < +\infty, \\ \alpha, & \text{if } -\infty < z < 0; \end{cases} \qquad \underline{\delta}_{\oplus}(z) = -\overline{\delta}_{\oplus}(-z);$$

$$\overline{\delta}_{\otimes}(z) = \lvert z \rvert \cdot 2^{-e_{\min}}; \qquad \underline{\delta}_{\otimes}(z) = -\overline{\delta}_{\otimes}(z);$$

$$\overline{\delta}_{\oslash}(z) = \lvert z \rvert \otimes f_{\max}; \qquad \underline{\delta}_{\oslash}(z) = -\overline{\delta}_{\oslash}(z).$$

- ☐ 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* **add** *E* | *E* **sub** *E* | *E* **mul** *E* | *E* **div** *E*
| *E* **==** *E* | *E* **!=** *E* | *E* **>** *E* | *E* **>=** *E*
| **(float**) *E* | **(double)** *E* | *Var* | *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

```
/* 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(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.
double(64):r in 1.342177280e+08 .. 1.342177280e+08
```

Selected experimental results (gcc/solaris/sparc)

Programs	Expected results	**Eclipse**	**FPSE**
[Goldberg 91] **2.0e-30 + 1.0e30 -1.0e30 - 1.0e-30**	**single: -1.000000003e-30** **double: -1.0e-30**	clpr: +0.0, clpq: $+10^{-30}$ ic: [-1.0e-30, 140737488355328]	**single: -1.000000003e-30** **double: -1.0e-30**
[Goldberg 91] **D == B^2 - 4AC** **A:=1.22 , B=3.34, D=+0.0**	single: 2.2859835624694824 double: 2.2859836065573770	clpr: 2.2859836065573771 clpq :27889/12200=2.285... ic: [2.2859836065573766, 2.2859839065573771]	single:[2.2859833240509033, 2.2859835624694824] double: [2.2859836065573766, 2.2859836065573770]
X < 1.0e4, **T_1= X +1.0e12,** **T_1 >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_1 == 1.e12**	single: [1.4012984643248171e-45, 3.2767998046875000e+04] double: [4.9406564584124654e-324, 6.1035156250000000e-05]	clpr,clpq : infeasible ic: infeasible	single: [1.4012984643248171e-45, 3.2768000000000000e+04] double: [4.9406564584124654e-324, 6.1035156250000000e-05]
power.c (X=10, Y = -40) **84 constraints**	single: +0.0 double: 1.000000000001e-40	clpr: +0.0, clpq: $+10^{-40}$ ic: [9.99999e-41, 1.0000000e-40]	single: +0.0 double: 1.000000000001e-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_1>100,X_2=100 -X_1,X_3=X_2-50,X_3 > 50.**	infeasible	clpr,clpq: infeasible ic: infeasible	infeasible

Experimental results with FPSE

EXPERIMENTAL RESULTS FOR `dichotomic()` (TIMEOUT = 30 MIN)

#	NbC	NbV	Global results		On the solution path				ULP Max		Speedup factor
			NbE	NbD	NbV	NbE	NbD	%	w/o	w/	
1	17	12	62	17,515	12	1	864	20.2	0.142	0.080	1.775
2	31	22	3,948	484,128	22	0	0	0.00	12.326	3.536	3.486
3	45	32	461	102,522	32	3	1,174	9.15	3.969	0.872	4.552
4	59	42	544,377	9,208,097	42	0	0	0.00	timeout	847.778	∞
5	73	52	510	158,716	52	5	1,895	8.86	2.370	1.506	1.574
6	87	62	799	209,621	62	0	0	0.00	timeout	2.050	∞
7	101	72	494	87,934	72	7	2,625	8.77	6.087	0.983	6.192
8	115	82	timeout	timeout	timeout	timeout	timeout	0.00	timeout	timeout	∞
9	129	92	258	83,166	92	9	3,338	8.67	2.352	0.978	2.405
10	143	102	637	157,421	102	0	0	0.00	timeout	2.482	∞
11	157	112	224	73,702	112	11	4,034	8.57	2.471	0.724	3.413
12	171	122	635	153,318	122	0	0	0.00	4.924	2.642	1.864

The speedup due to ULP-Maximum does not depend on NbC or NbV!

EXPERIMENTAL RESULTS FOR `tcas_periodic_task_1Hz()`

#	NbC	NbV	Global results		On the solution path				ULP Max		Speedup factor
			NbE	NbD (M)	NbV	NbE	NbD (M)	%	w/o	w/	
1	157	191	5	765	191	1	11	0.28	1.200	1.212	0.99
2	152	191	1	45	191	1	45	1.07	3.261	3.313	0.98
3	152	191	1	45	191	1	45	1.07	3.688	3.715	0.99
4	152	191	4	753	191	0	0	0.00	0.039	0.032	1.22
5	152	191	4	753	191	0	0	0.00	0.041	0.037	1.11
6	157	191	4	955	191	0	0	0.00	0.060	0.048	1.25
7	157	191	4	955	191	0	0	0.00	0.071	0.078	0.91
8	157	191	25	1,884	191	20	1,884	2.20	0.046	0.046	1.00
9	157	191	25	1,884	191	20	1,884	2.20	0.369	0.382	0.97
10	157	191	25	1,884	191	20	1,884	2.20	0.068	0.068	1.00
11	157	191	25	1,884	191	20	1,884	2.20	0.706	0.698	1.01
12	152	191	25	1,884	191	20	1,884	2.20	0.029	0.027	1.05
13	152	191	25	1,884	191	20	1,884	2.20	0.027	0.029	0.93
14	157	191	3	387	191	1	10	0.24	0.076	0.030	2.53
15	157	191	3	395	191	0	0	0.00	0.081	0.039	0.93
16	157	191	1	43	191	1	43	1.01	0.071	0.076	0.93
17	157	191	3	387	191	1	10	0.24	0.074	0.032	2.31
18	157	191	3	395	191	0	0	0.00	0.083	0.040	2.08
19	157	191	1	43	191	1	43	1.01	0.075	0.076	0.99
20	152	191	1	43	191	1	43	1.01	0.079	0.079	1.00
21	152	191	1	43	191	1	43	1.01	0.075	0.075	1.00
22	152	191	8	521	191	6	144	0.56	0.077	0.033	2.33
23	152	191	8	521	191	6	144	0.56	0.077	0.033	2.33
24	157	191	2	477	191	0	0	0.00	0.079	0.031	2.55
25	157	191	2	477	191	0	0	0.00	0.074	0.032	2.31
26	152	191	1	43	191	1	43	1.01	0.075	0.077	0.97
27	152	191	1	43	191	1	43	1.01	0.078	0.077	1.01

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 !