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 V^{3}F Project :

Validation & Vérification 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


```
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:

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, 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

significand (23, 52 bits or extended)

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

sign (1 bit)

exponent (8, 11 bits or extended)

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

⌘ $e = e_{max}$: +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 ($X_f + \varepsilon == X_f$), 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*) :
$(Path, State, PC)$ where

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

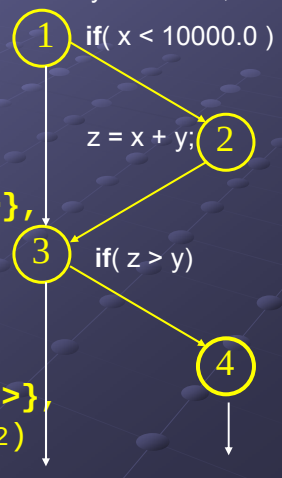
Symbolic state : features

- `(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
  then $\forall X \in S_{PC}$, `Path` is activated by X
- When $S_{PC}=\varnothing$ then `Path` is non-feasible

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

(Path,State,PC) : examples

```
float foo( float x) {
float y = 1.0e12 ;
```



`(1,{<x,X>,<y,1.0e12>,<z,⊥>},true)`

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

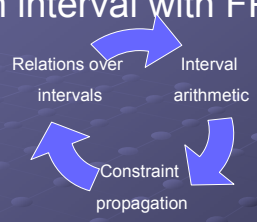
`(1→2→3→4,`
`{<x,X>,<y,1.0e12>,<z,X add 1.0e12>},`
`X<10000.0, T := X add 1.0e12, T > 1.0e12)`

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_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 add y
                          { v := t add 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 [Colmerauer 94]
- Numerica [Van Hentenryck 97]
- ECLIPSE(IC) [Wallace ... 97]
- Realpaver [Granvilliers 98]
- ...

Interval Arithmetic over the reals:

I_x = [a,b] and I_y = [c,d] then

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

...

Our approach: floating-point projections

⌘Direct and indirect projections for the assignment:

[r := a add b] leads to
- proj(r, r:= a add b) (direct)
- proj(a, r:= a add b) (1st indirect)
- proj(b,r := a add 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

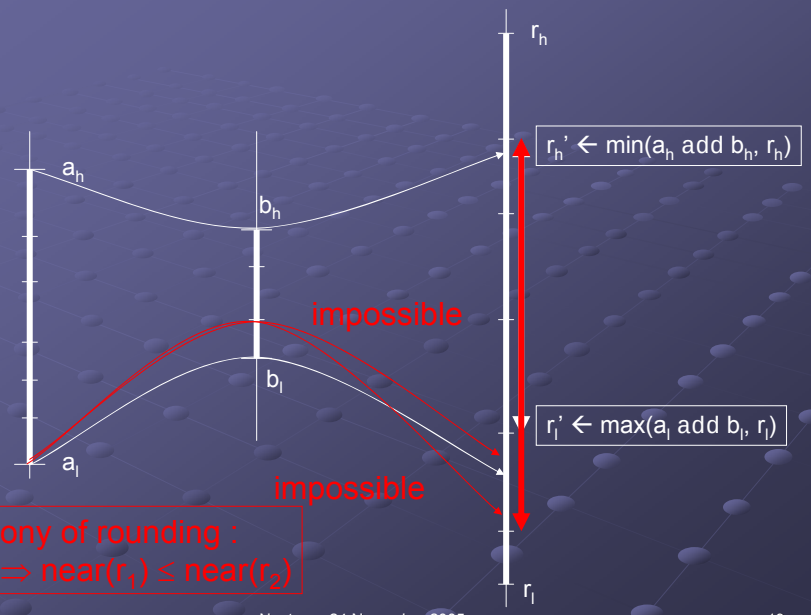
[r := a add b] $[r_l',r_h'] \leftarrow [a_l \text{ add } b_l, a_h \text{ add } b_h] \cap [r_l,r_h]$

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

...

Recall that [x add y] denotes **near(**x + y)

Ex: Direct projection [r := a add b]



Monotony of rounding :
$r_1 \leq r_2 \Rightarrow near(r_1) \leq near(r_2)$

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 add b]

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

1st indirect projection of [r := a subs b]

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

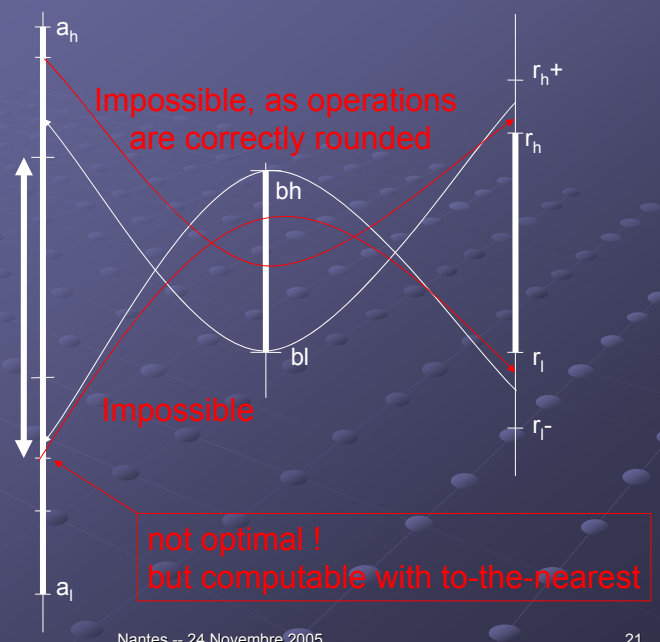
2nd indirect projection of [r := a subs b]

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

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

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

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



Handling comparisons and conversions

<u>Comparisons (1st proj) :</u>

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

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

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

<u>Floating-point conversions:</u>

when [r := (float)a]

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

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

NaNs (for example INF – 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 as the INKA tool
  B. Botella : full design and first implementation
  Me : constraint propagation engine and improvments on fp projection functions

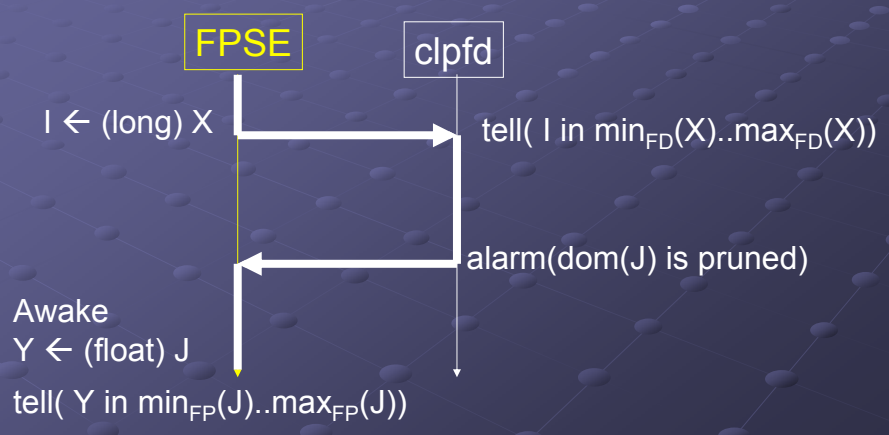
- Handles C computations for gcc/solaris/sparc
and visual/xp/intel (unsound w.r.t. the stack of 80-bits registers)

- 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



An example (extracted from www.astree.ens.fr/)

```
/* 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 1.342177280000000000e+08 .. 1.342177280000000000e+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: [1.401298464324817 1e-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

Problems: weakness of fp_constraints to prune the domains

example adapted from [KPV TACAS'03] :

```
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('-',X1,Y1,X2),
        flottant:affect('-',X2,Y1,T1),
        flottant:affect(const('0.0'),C1),
        flottant:affect('>',T1,C1,R),  R = 1,
        solveur:solve(E),
        flottant:fprint([X,Y]).

| ?- test12(float(32)).

float(32):x in -3.402823466385288598e+38 .. Inf
float(32):y in -Inf .. 3.402823466385288598e+38
```


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

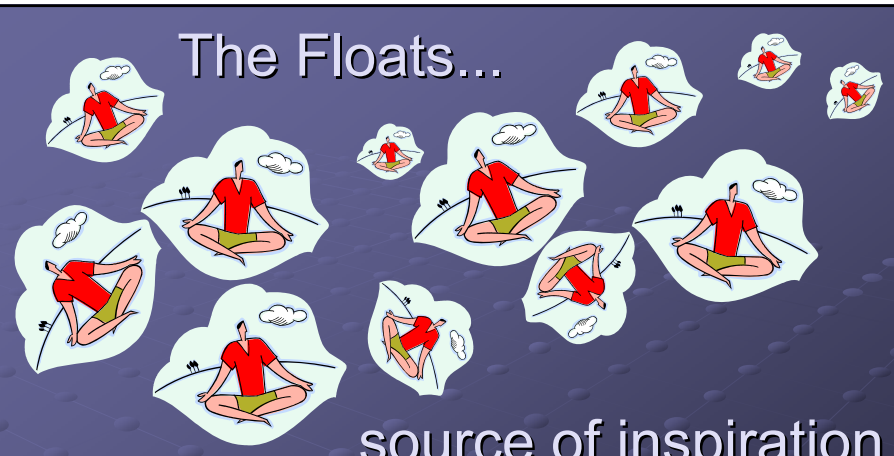
Extensions: NaNs

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

Problems: Improving the pruning capacity of projection functions

Labelling process (heuristics)

The Floats...



...source of inspiration ?

1. B. Botella, A. Gotlieb, C. Michel
«Symbolic execution of floating-point computations»
Software Testing and Verification Journal, to appear in 2006
2. B. Botella, A. Gotlieb
«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

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

FPSE and FPLIB with automatic decomposition are incomparable

[exp := 1.0e12f + 32000.0f + 32000.0f with to-the-nearest rounding mode]

In FPSE:

```
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 */
```

```
float(32):exp in 9.999999959040000000e+11 ..
                 9.999999959040000000e+11
```

In FPLIB:

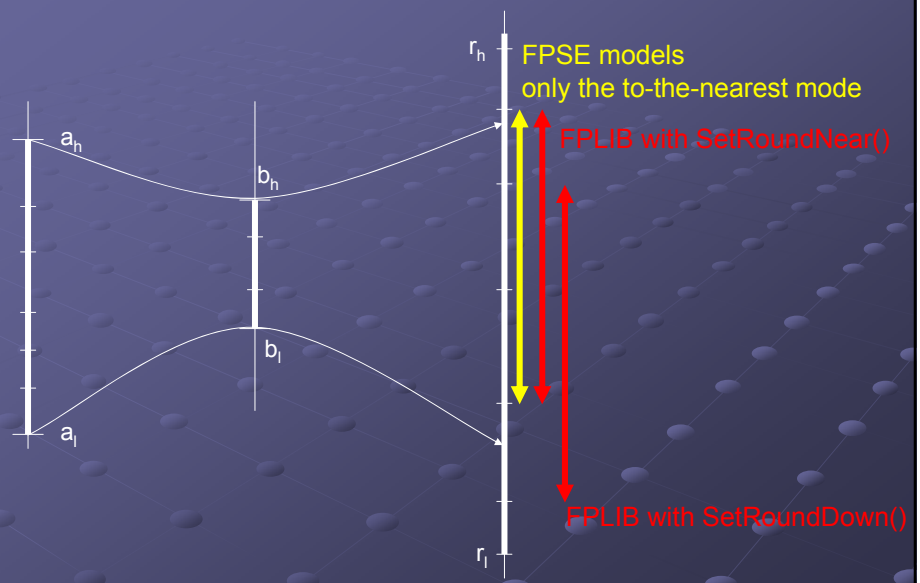
```
int main(void) {
int nbsols = 0;   Fpc_Csp CSP;
Fpc_Variable a(CSP, "a", FPC_FLOAT);
  ...
Fpc_Model model(CSP);
model.add(a = 1.0e12);
model.add(b = 32000.0);
model.add(c = 32000.0);
model.add(exp = a + b + c);
model.extract();
Fpc_Solver solver(model);
if( ! solver.TwoB(0.0) ) printf("pas de solution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

```
% make P=deco ; deco  /* Linux pal.irisa.fr 2.6.10-
    1.12_FC2smp # i686 i686 i386 GNU/Linux
```

```
exp = [ 1.000000061440000000e+12,
        1.000000061440000000e+12]f
```

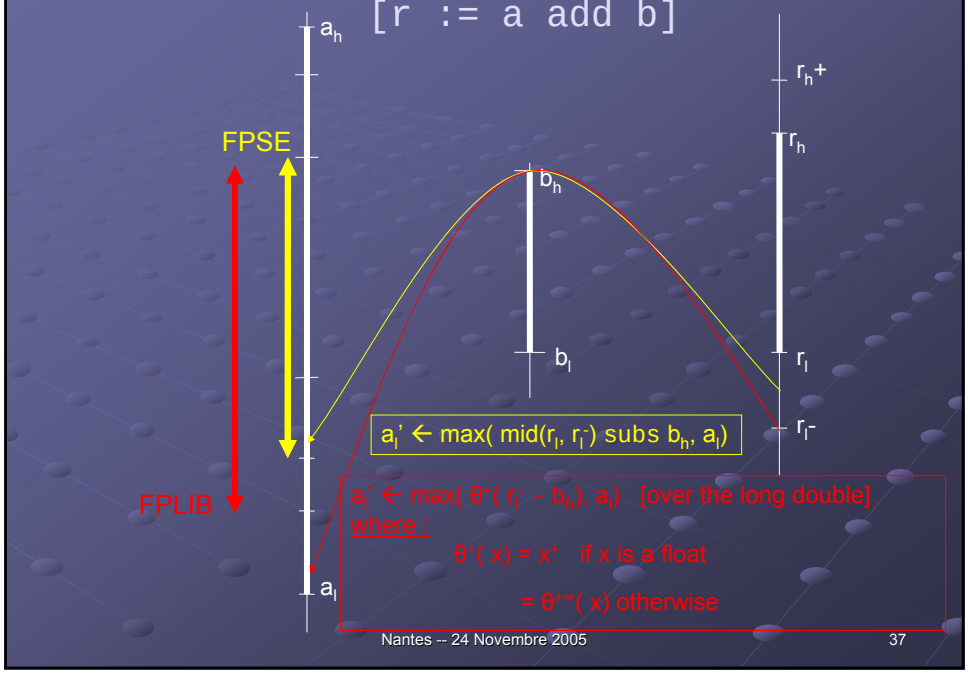

direct projection of a basic operation

`[r := a add b]`



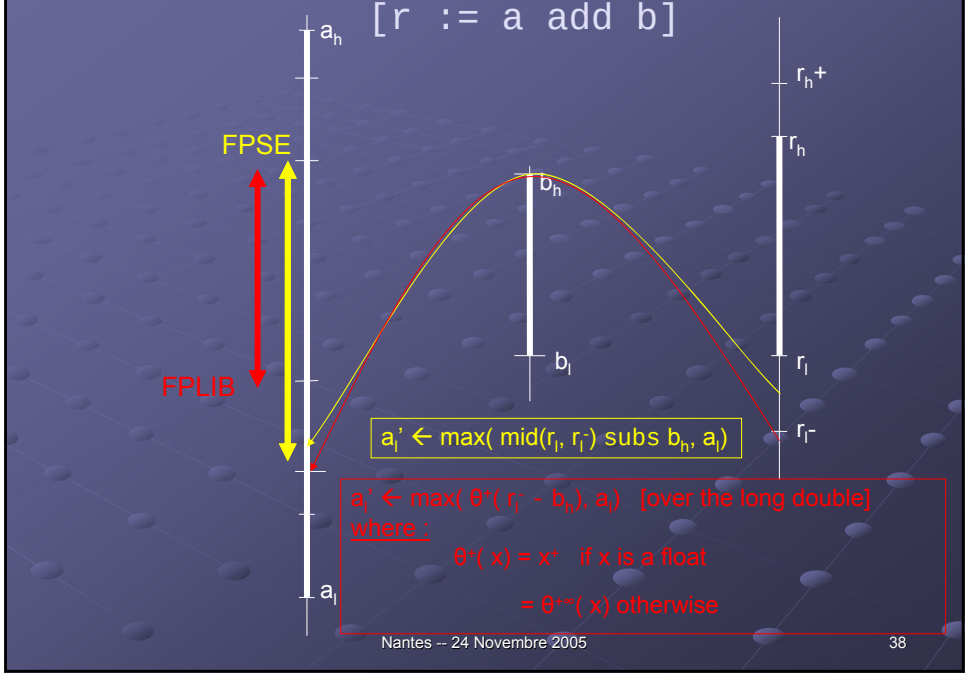
inverse projection of a basic operation

[r := a add b]



inverse projection of a basic operation

[r := a add b]



Very first experimental comparison

Programs	Expected results	FPLIB	FPSE
Y = log X,Y = sin X, ...	idem	idem	no
g1, g2, power9,...	idem	idem	idem
X < 1.0e4, T_1= X +1.0e12, T_1 >1.0e12	single: infeasible path double: [6.103e-5, 9.999e3]	single: infeasible path double: [6.103e-5, 9.999e3	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]	single: [1.4012984643248171e-45, 3.2768000000000000e+04] double: [4.9406564584124654e-324, 6.1035156250000000e-05]	single: [1.4012984643248171e-45, 3.2768000000000000e+04] double: [4.9406564584124654e-324, 6.1035156250000000e-05]
[Goldberg 91] D == B^2 - 4AC A:=1.22f , B=3.34f, D=+0.0f	single: 2.2859835624694824	c = 2.2859833240509033 with SetFloatPrecision(); c = 2.2859835624694824 with SetLDoublePrecision(); ???	single:c [2.2859833240509033, 2.2859835624694824]
trap computes the area of sin between 0 and Pi 38 constraints	area6 = 1.9337656497955322 ??	area6 = 1.933765649795532227 by using direct projection over a selected path Pi in [3.141592, 3.141593] 4 solutions	?? ??

Summary

FPLIB :

+ - transcendentals
- automatic decomposition
- rounding mode / precision

- documentation

source code ?

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

4 projection functions

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

$X \in$	$[-\infty,+\infty]$	$[0,+\infty]$	$[0, 1]$	$[0.56, 1]$	$[0.56, 0.57]$
$Y \in$	$[-\infty,+\infty]$	$[-\infty, 0]$	$[-1, 0]$	$[-1, -0.56]$	$[-0.57, -0.56]$

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