On Suspicious Intervals for Floating-Point Number Programs

Michel RUEHER

joined work with

Hélène Collavizza, Claude Michel, Olivier Ponsini, Mohammed Said Belaid

University of Nice Sophia-Antipolis I3S – CNRS

France

Dagstuhl Seminar 14351

24 - 29 August 2014

Context

- Embedded Systems (Anti-lock Braking System controller, ...) rely more and more on floating-point computations
- C language is widely used for such applications (often C code generated from a Simulink model)
- **Floats** $\rightarrow$ an additional source of errors

Problems with floating-point numbers

Rounding $\rightsquigarrow$ *Counter-intuitive properties*

$\rightarrow$ Arithmetic operators are neither associative nor distributive

$\rightarrow$ Reasoning with absorption and cancellation

Examples (in simple precision, binary representation):

- Absorption : $10^7 + 0.5 = 10^7$
- Cancellation : $((1 - 10^{-7}) - 1) * 10^7 = -1.192...(\neq -1)$
- $(10000001 - 10^7) + 0.5 \neq 10000001 - (10^7 + 0.5)$
- $0.1 = (0.000110011001100\ldots)$

Programs with floating-point numbers

Programs are run on the floats but:

- Specification, properties of programs
  $\rightsquigarrow$ Users are reasoning with real numbers
- Programs are often written with the semantics of real numbers “in mind”
- Differences between computations over real numbers and computations over the floats

  $\rightarrow$ *Execution problems on programs with floats*

Verifying programs with floats

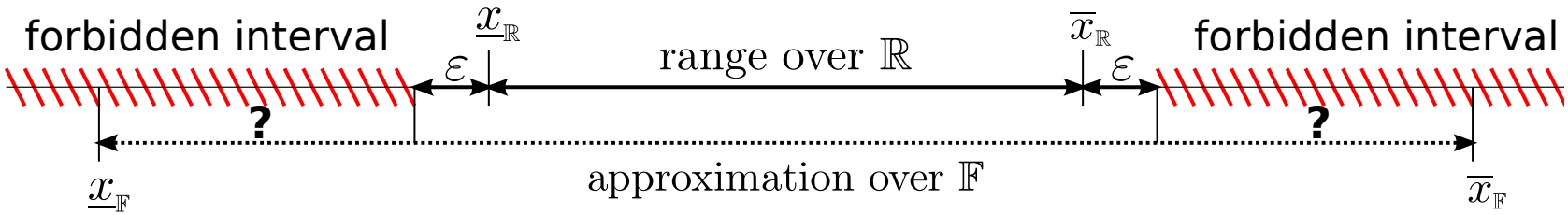
Abstract Interpretation: good scalability for estimating rounding errors but *over-approximation*

→ *false alarms*

→ *totally inappropriate behaviours of a program may be dreaded but the developer does not know whether these behaviours will actually occur !*

Proposed approach: combining AI and CP for approximating suscipicious intervals and finding counter examples

Suspicious Intervals



Suspicious intervals for x : $[\underline{\mathbf{x}}_F, \underline{\mathbf{x}}_R - \varepsilon]$ and $[\overline{\mathbf{x}}_R + \varepsilon, \overline{\mathbf{x}}_F]$

Tolerance specified by the user :ε

Question: Can the program hit a forbidden zone over the floating-point numbers?

Proposed approach (1)

"Forward" propagation

Computing the suspicious interval of x

$\rightarrow$ approximate the domain of x over the reals by $[\underline{\mathbf{x}}_R, \overline{\mathbf{x}}_R]$

$\rightarrow$ approximate domain of x over the floats by $[\underline{\mathbf{x}}_F, \overline{\mathbf{x}}_F]$

"Backward " propagation

Computing test-cases inside a suspicious interval of x

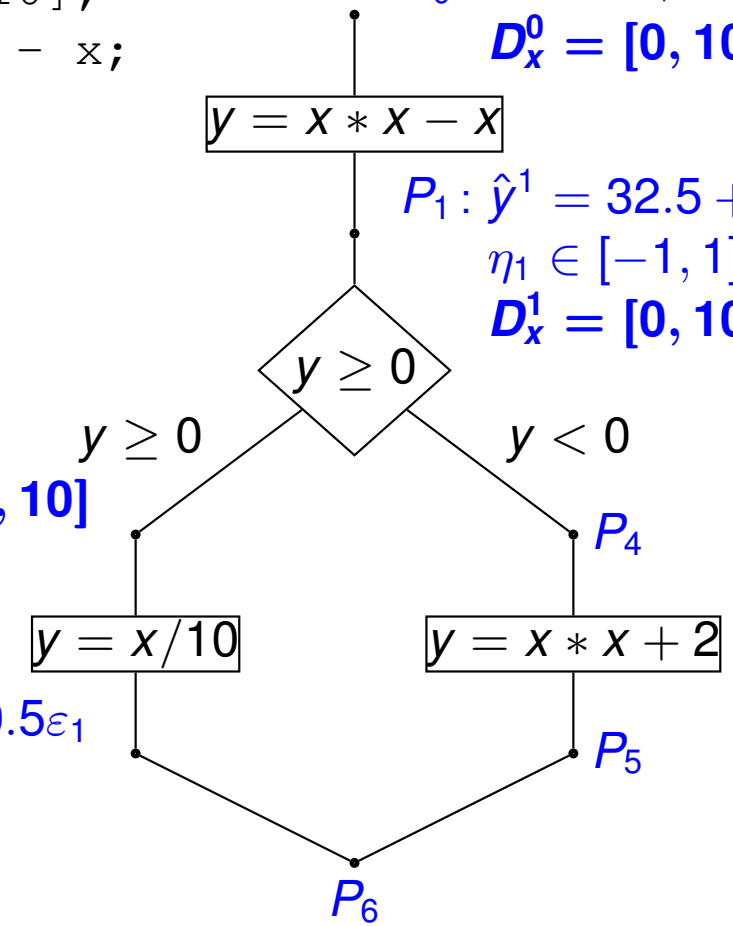
$\rightarrow$ Solving a bounded-model checking problem with domain of x restricted to $[\underline{\mathbf{x}}_F, \underline{\mathbf{x}}_R - \varepsilon]$ or $[\overline{\mathbf{x}}_R + \varepsilon, \overline{\mathbf{x}}_F]$

Motivating example

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
```

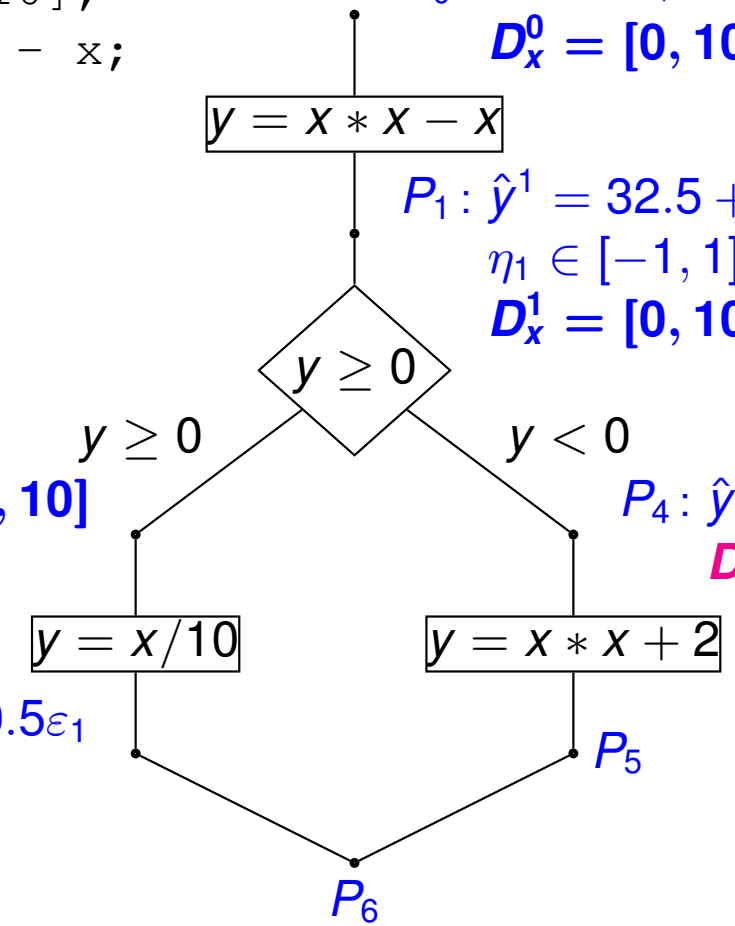
Example 1: Abstract Interpretation (zonotopes)

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
```



Example 1: Abstract Interpretation (zonotopes)

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
```



$P_0 : \hat{x}^0 = 5 + 5\varepsilon_1 \quad \varepsilon_1 \in [-1, 1]$
$D_x^0 = [0, 10]$

$P_1 : \hat{y}^1 = 32.5 + 45\varepsilon_1 + 12.5\eta_1$
$\eta_1 \in [-1, 1]$
$D_x^1 = [0, 10] \quad D_y^1 = [-10, 90]$

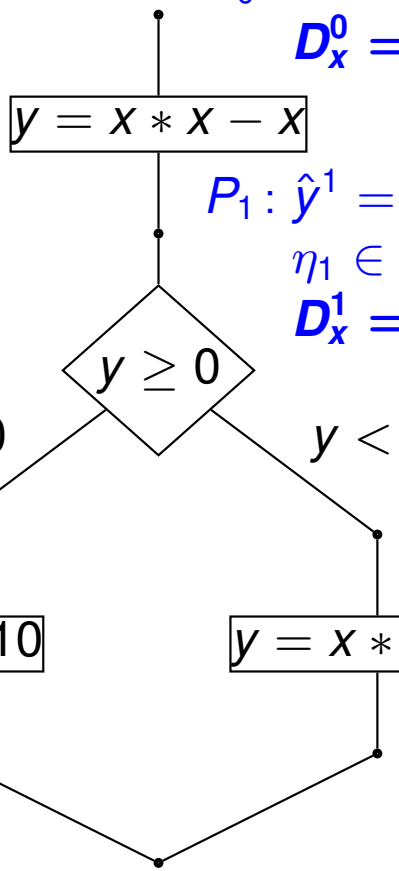
$P_2 : \hat{y}^2 = \hat{y}^1 \quad D_x^2 = [0, 10]$
$D_y^2 = [0, 90]$

$P_3 : \hat{y}^3 = 0.5 + 0.5\varepsilon_1$
$D_y^3 = [0, 1]$

$P_4 : \hat{y}^4 = \hat{y}^1 \quad D_x^4 = [0, 10]$
$D_y^4 = [-10, 0[$

Example 1: Abstract Interpretation (zonotopes)

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
```



$P_0 : \hat{x}^0 = 5 + 5\varepsilon_1 \quad \varepsilon_1 \in [-1, 1]$
$D_x^0 = [0, 10]$

$P_1 : \hat{y}^1 = 32.5 + 45\varepsilon_1 + 12.5\eta_1$
$\eta_1 \in [-1, 1]$
$D_x^1 = [0, 10] \quad D_y^1 = [-10, 90]$

$P_2 : \hat{y}^2 = \hat{y}^1 \quad D_x^2 = [0, 10]$
$D_y^2 = [0, 90]$

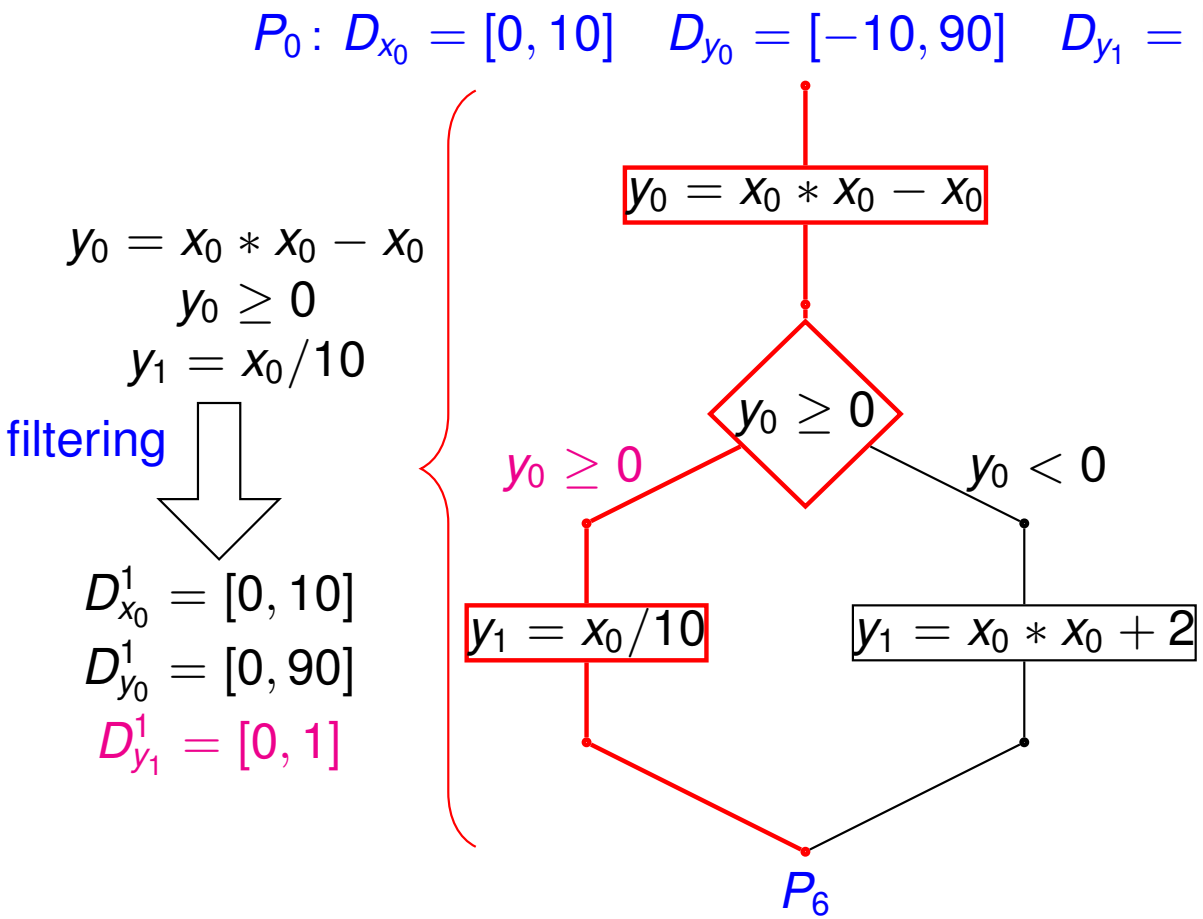
$P_4 : \hat{y}^4 = \hat{y}^1 \quad D_x^4 = [0, 10]$
$D_y^4 = [-10, 0[$

$P_3 : \hat{y}^3 = 0.5 + 0.5\varepsilon_1$
$D_y^3 = [0, 1]$

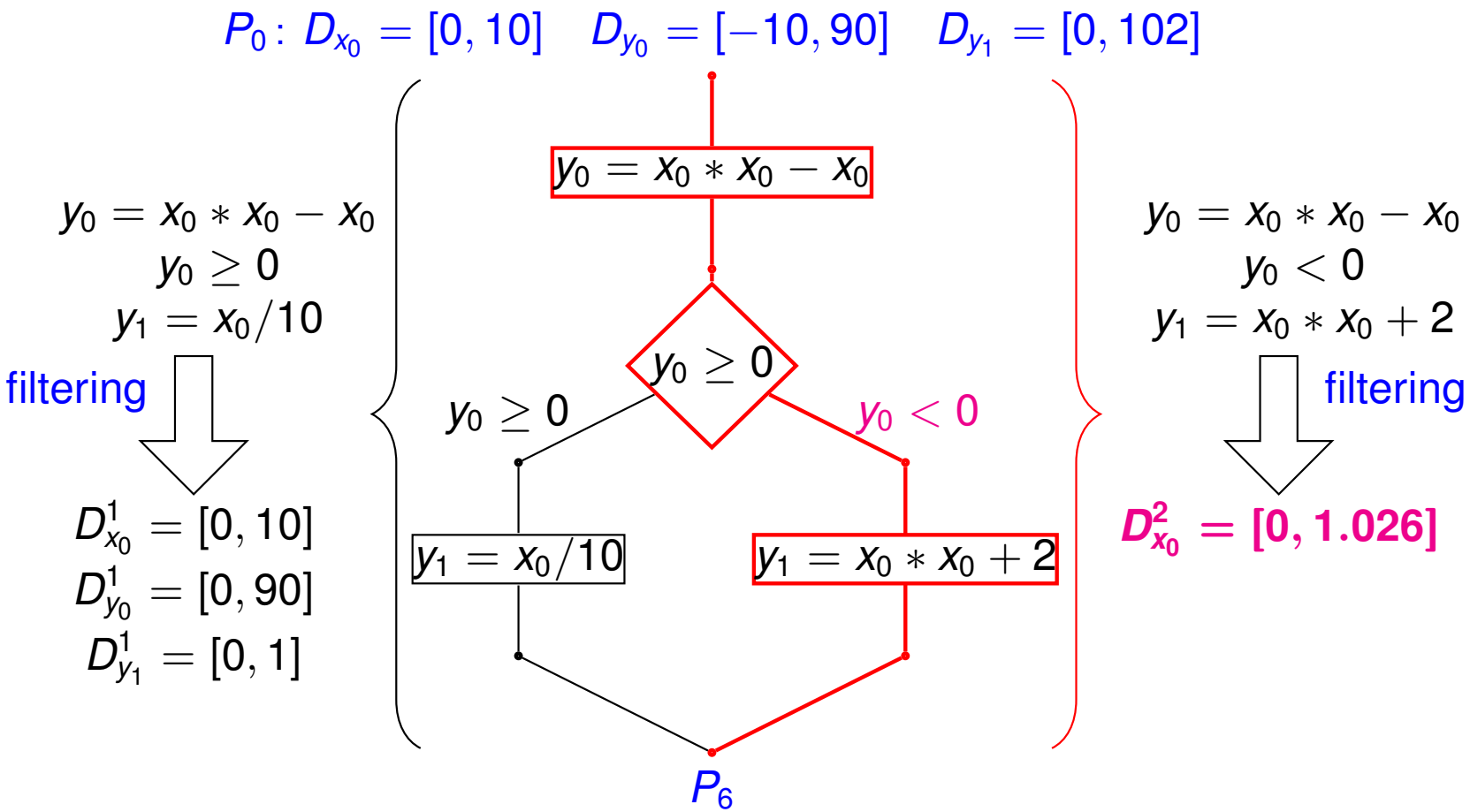
$P_5 : \hat{y}^5 = 39.5 + 50\varepsilon_1 + 12.5\eta_1$
$D_y^5 = [2, 102]$

$P_6 : \hat{y}^6 = \hat{y}^3 \cup \hat{y}^5 = 39.5 + 0.5\varepsilon_1 + 62\eta_2$
$\eta_2 \in [-1, 1]$
$D_y^6 = D_y^3 \cup D_y^5 = [0, 102]$

Example 1: Constraint Programming



Example 1: Constraint Programming



Example 1: Constraint Programming

$P_0 : D_{x_0} = [0, 10] \quad D_{y_0} = [-10, 90] \quad \mathbf{D_{y_1} = [0, 102]}$

$y_0 = x_0 * x_0 - x_0$

$y_0 \geq 0$

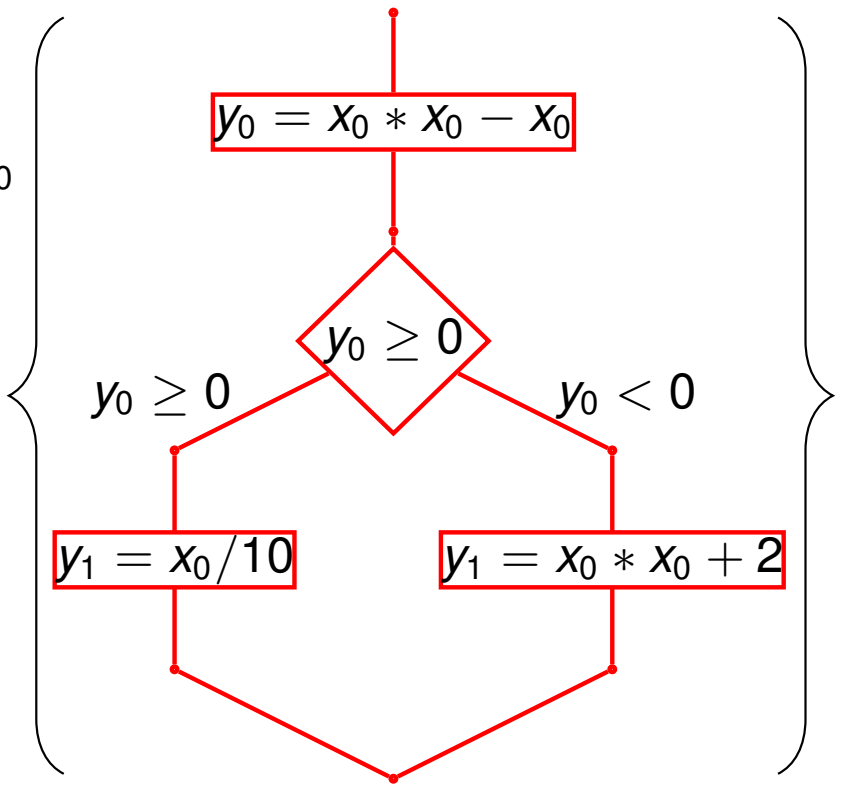
$y_1 = x_0/10$

filtering

$D^1_{x_0} = [0, 10]$

$D^1_{y_0} = [0, 90]$

$D^1_{y_1} = [0, 1]$



$y_0 = x_0 * x_0 - x_0$

$y_0 < 0$

$y_1 = x_0 * x_0 + 2$

filtering

$D^2_{x_0} = [0, 1.026]$

$D^2_{y_0} = [-0.257, 0]$

$\mathbf{D^2_{y_1} = [2, 3.027]}$

$P_6 : D^3_{y_1} = D^1_{y_1} \cup D^2_{y_1} = \mathbf{[0, 3.027]}$

Proposed approach (1)

"Forward" propagation

Computing the suspicious interval of x

$\rightarrow$ approximate the domain of x over the reals by $[\underline{\mathbf{x}}_R, \overline{\mathbf{x}}_R]$

$\rightarrow$ approximate domain of x over the floats by $[\underline{\mathbf{x}}_F, \overline{\mathbf{x}}_F]$

"Backward " propagation

Computing test-cases inside a suspicious interval of x

$\rightarrow$ Solving a bounded-model checking problem with domain of x restricted to $[\underline{\mathbf{x}}_F, \underline{\mathbf{x}}_R - \varepsilon]$ or $[\overline{\mathbf{x}}_R + \varepsilon, \overline{\mathbf{x}}_F]$

Proposed approach (2)

Computing approximations: RAICP, a hybrid system combining :

- FLUCTUAT : AI system based on zonotopes
- FPCS: solver over floating-point constraints

→ a *symbolic execution* approach combining interval propagation with explicit search for floating-point constraint problems

Computing test-cases: CPBPV_FP:

→ adaptation of CPBPV for generating test cases hitting a suspicious interval in programs with floating-point computations

→ based on FPCS

FPCS: Solver over floating-point constraints

→ Symbolic execution approach combining interval propagation with explicit search for satisfiable floating-point problems

→ Filtering techniques : 2B and 3B-consistency over the floats

- Projection functions for floats
- Handling of rounding modes

Techniques derived from CSP techniques over continuous domains

CSP over continuous domains – overall scheme

CP over continuous domains $\equiv$ a branch & prune process
$\rightarrow$ an iteration of two steps:

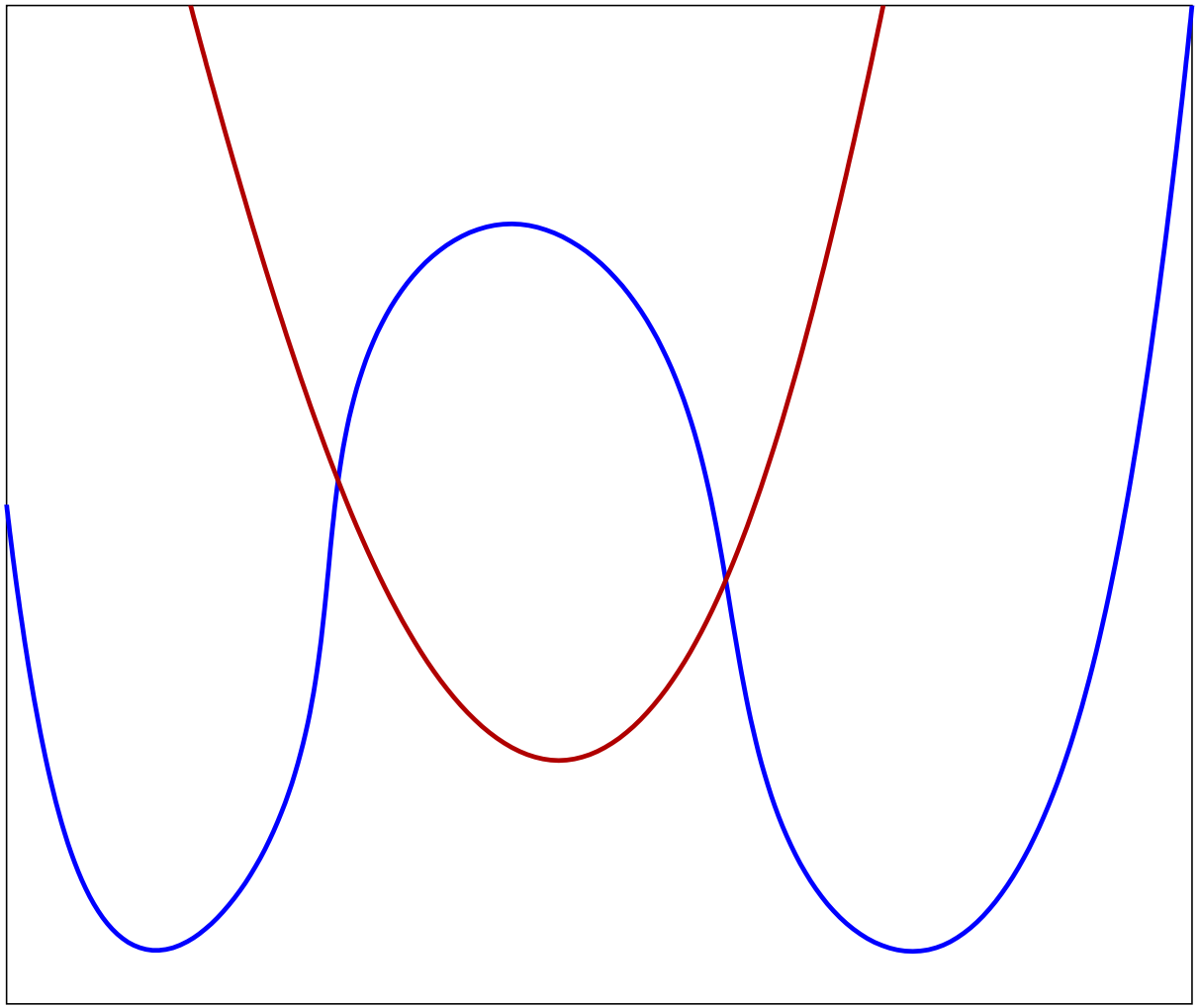
1. **Pruning the search space**
2. **Making a choice to generate two (or more) sub-problems**

Pruning step $\rightarrow$ reduces an interval when the upper bound or the lower bound does not satisfy some constraint

Branching step $\rightarrow$ splits the domain of some variable in two or more intervals

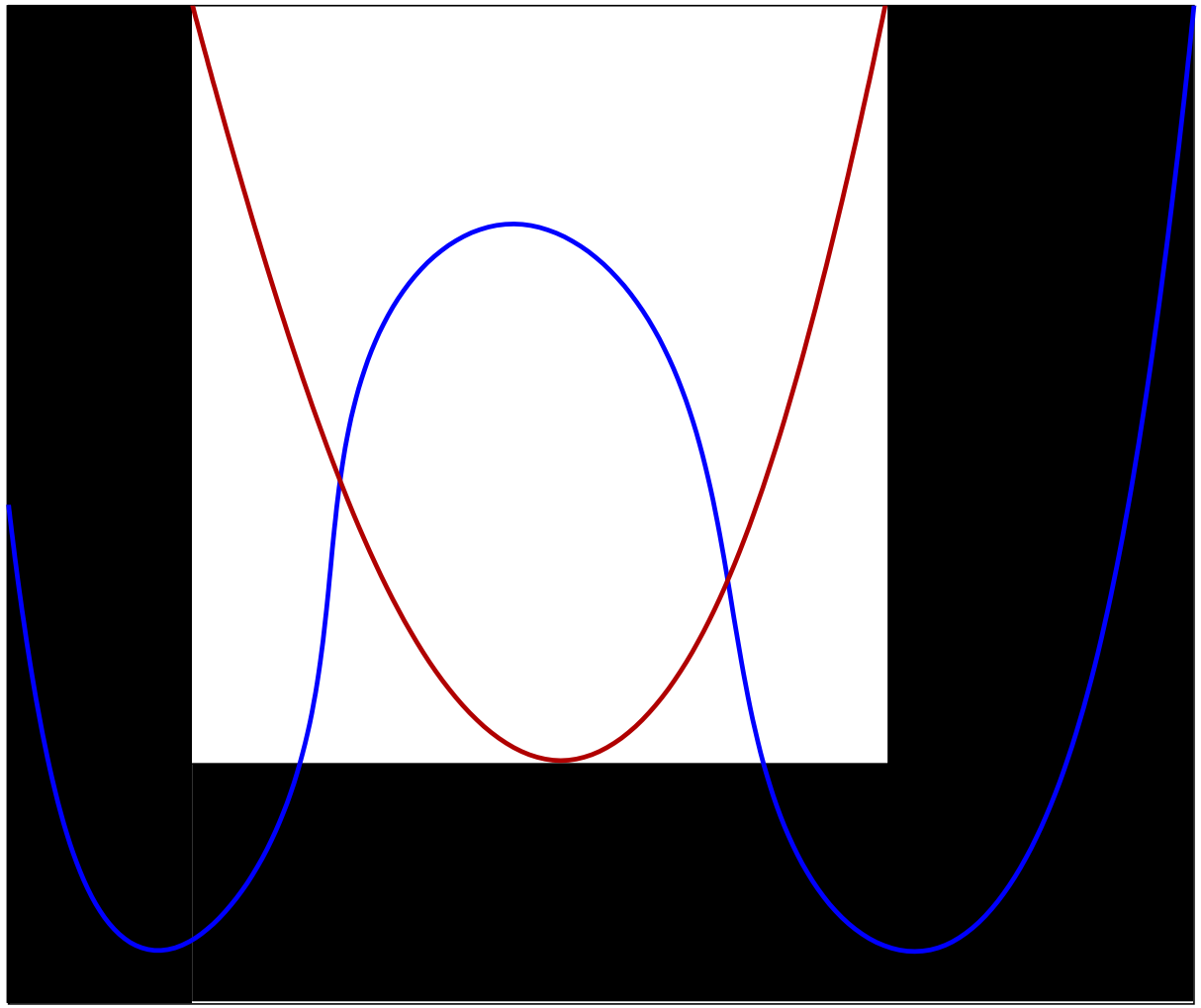
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



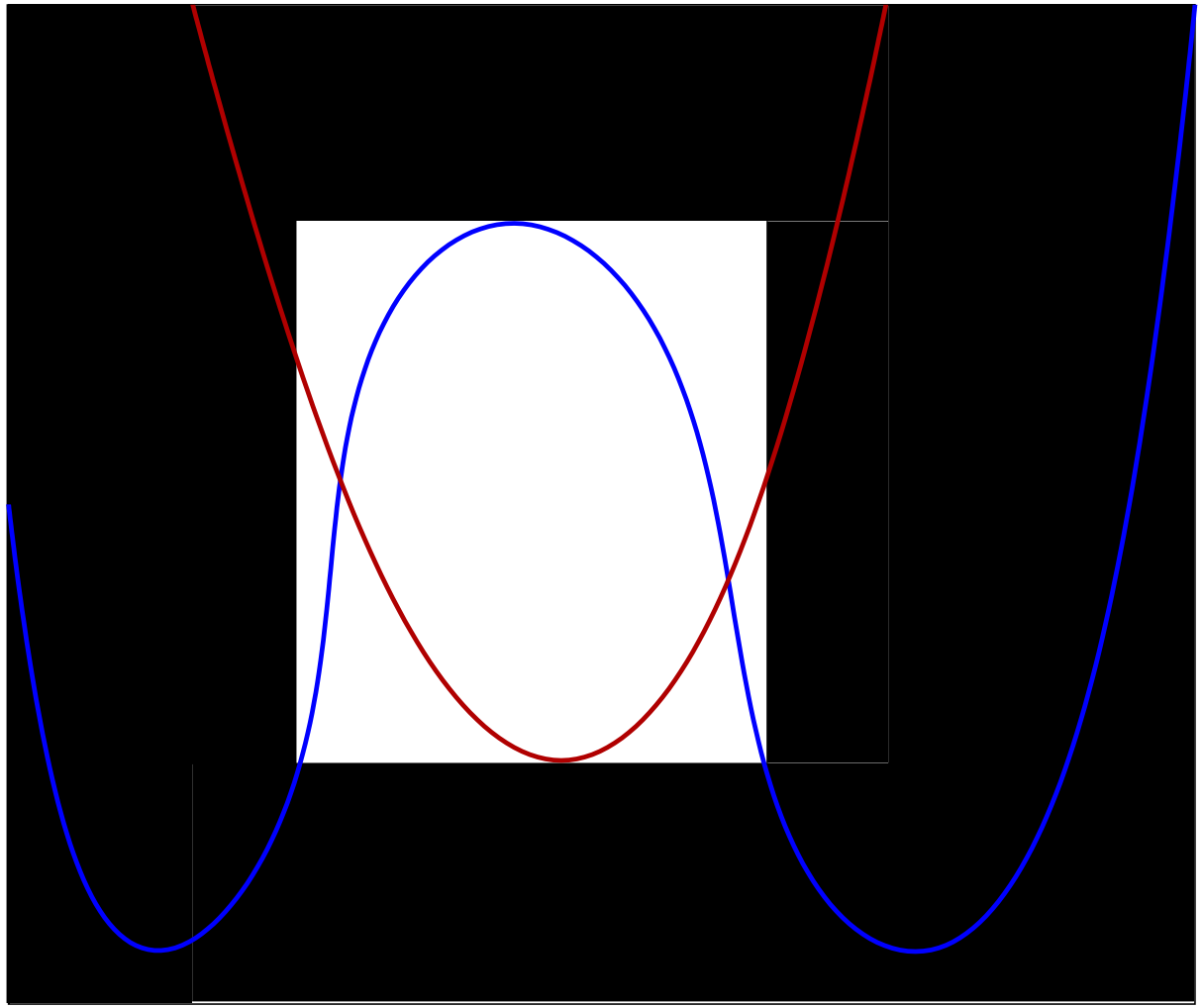
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



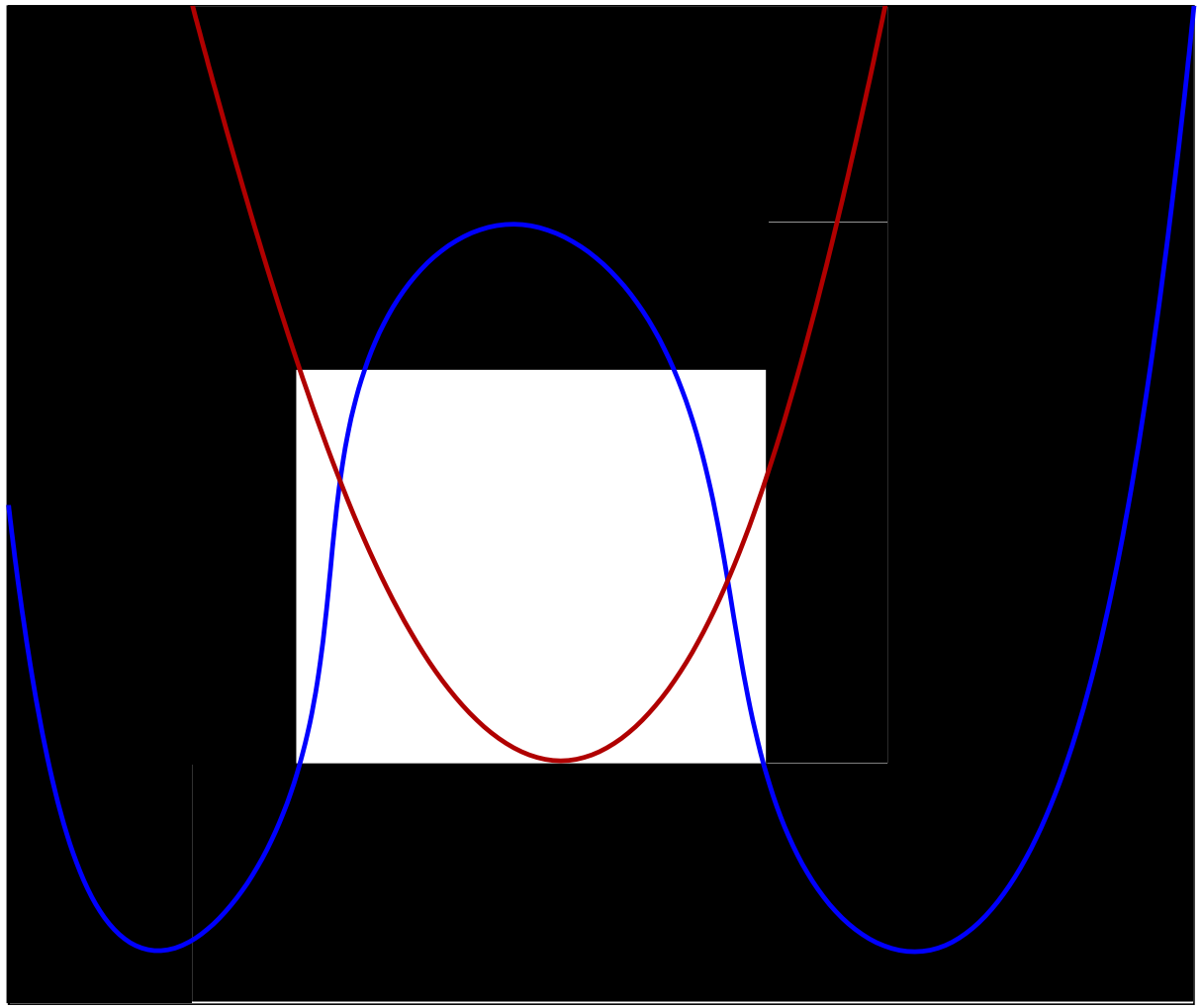
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



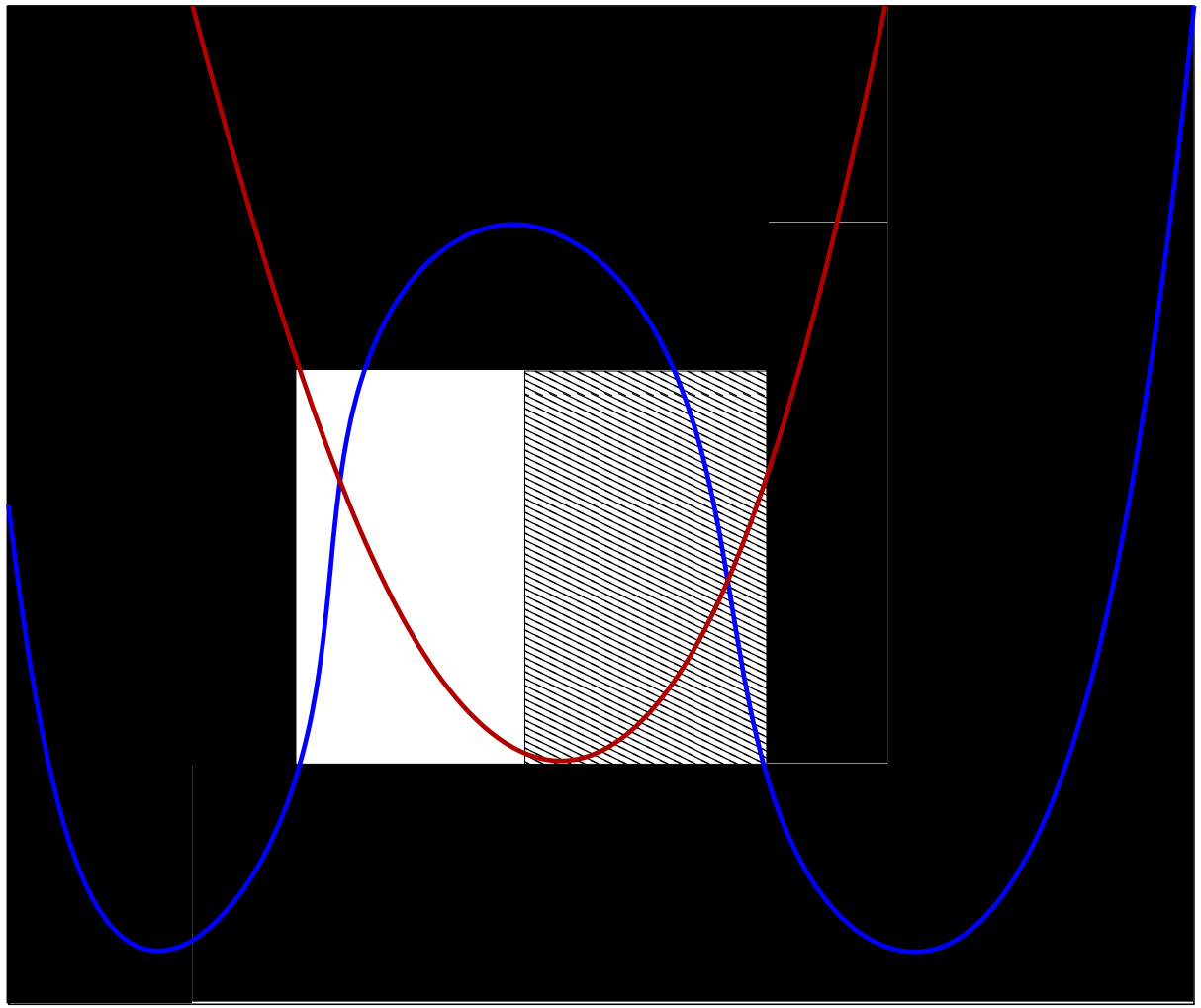
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



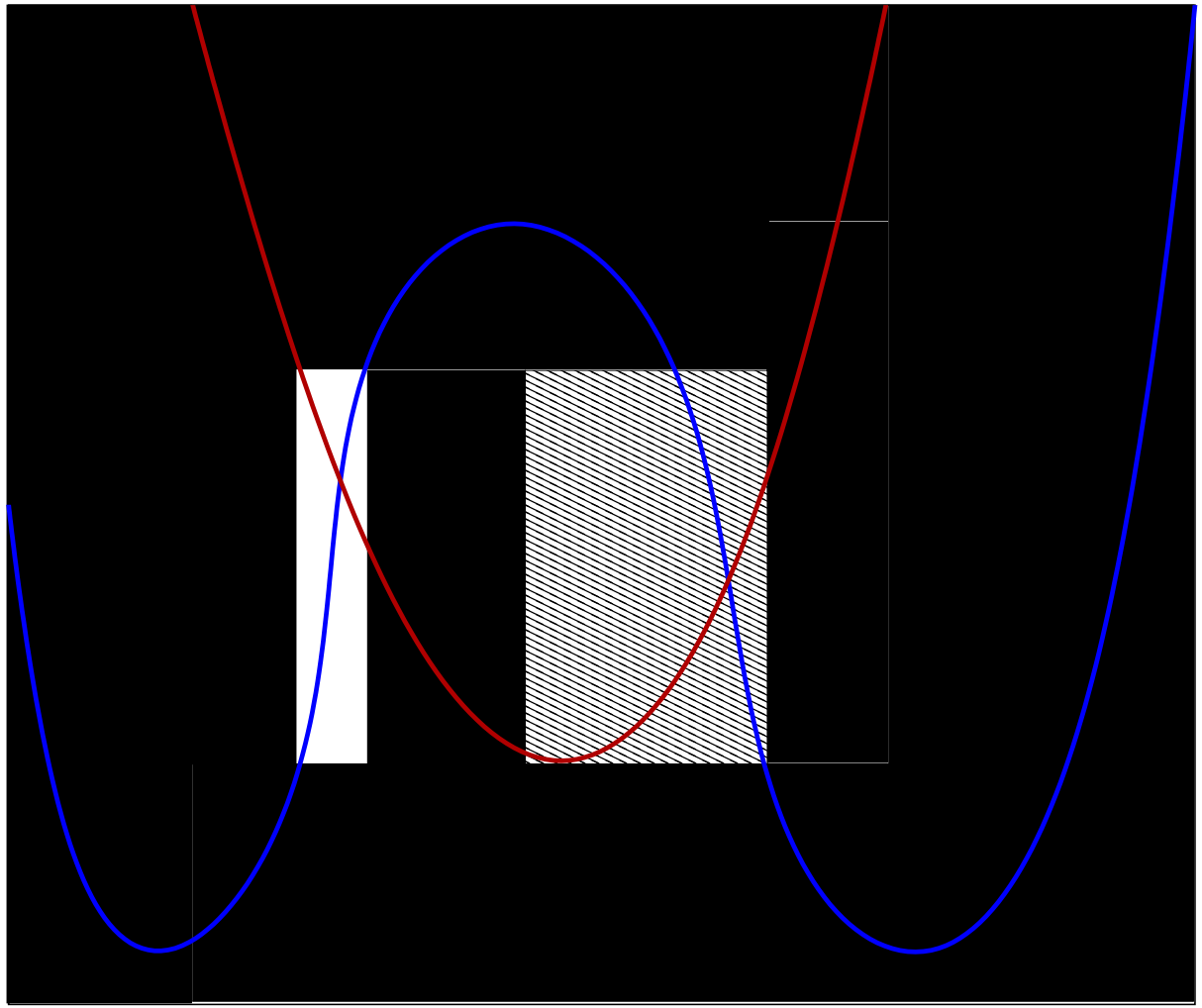
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



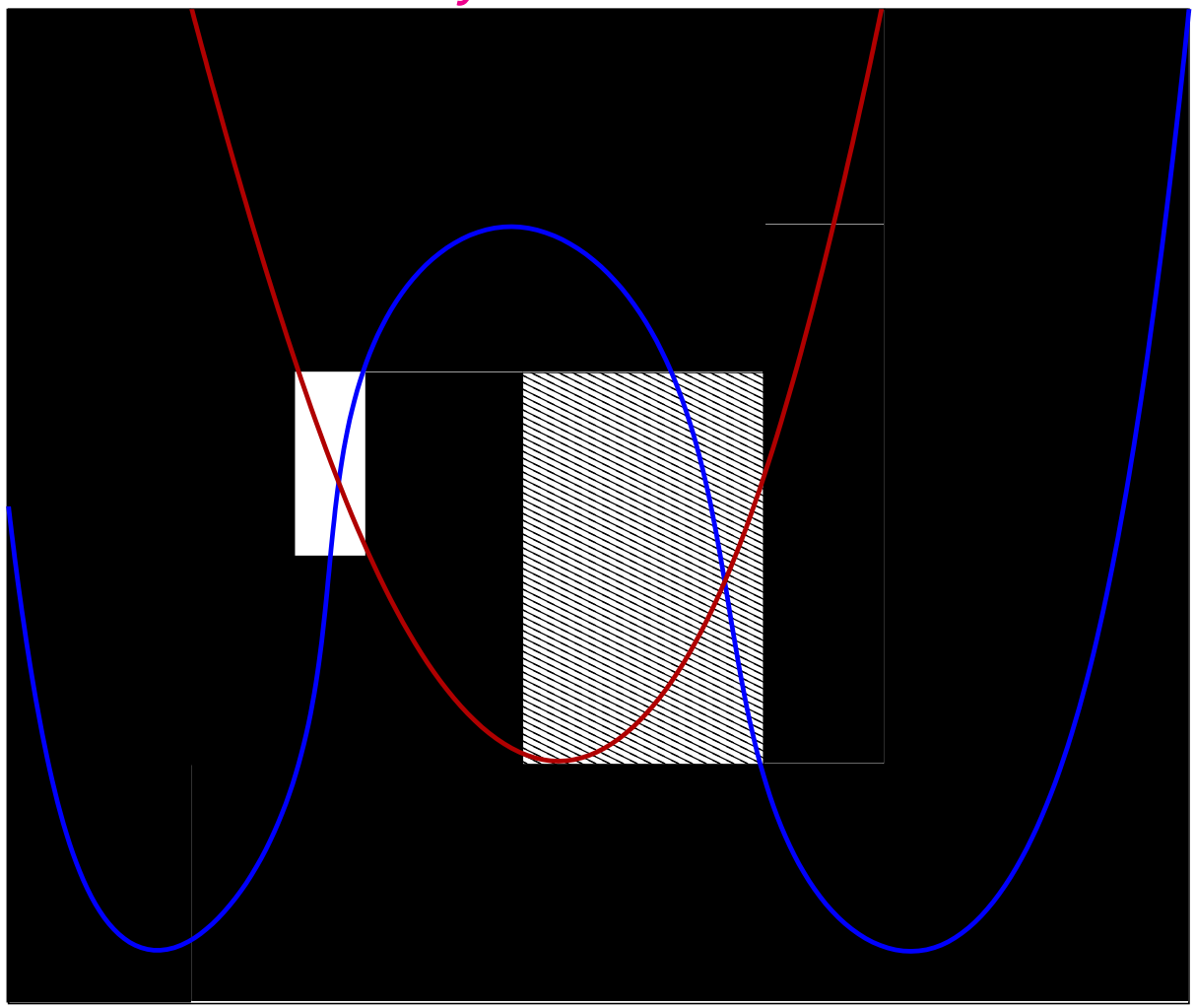
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



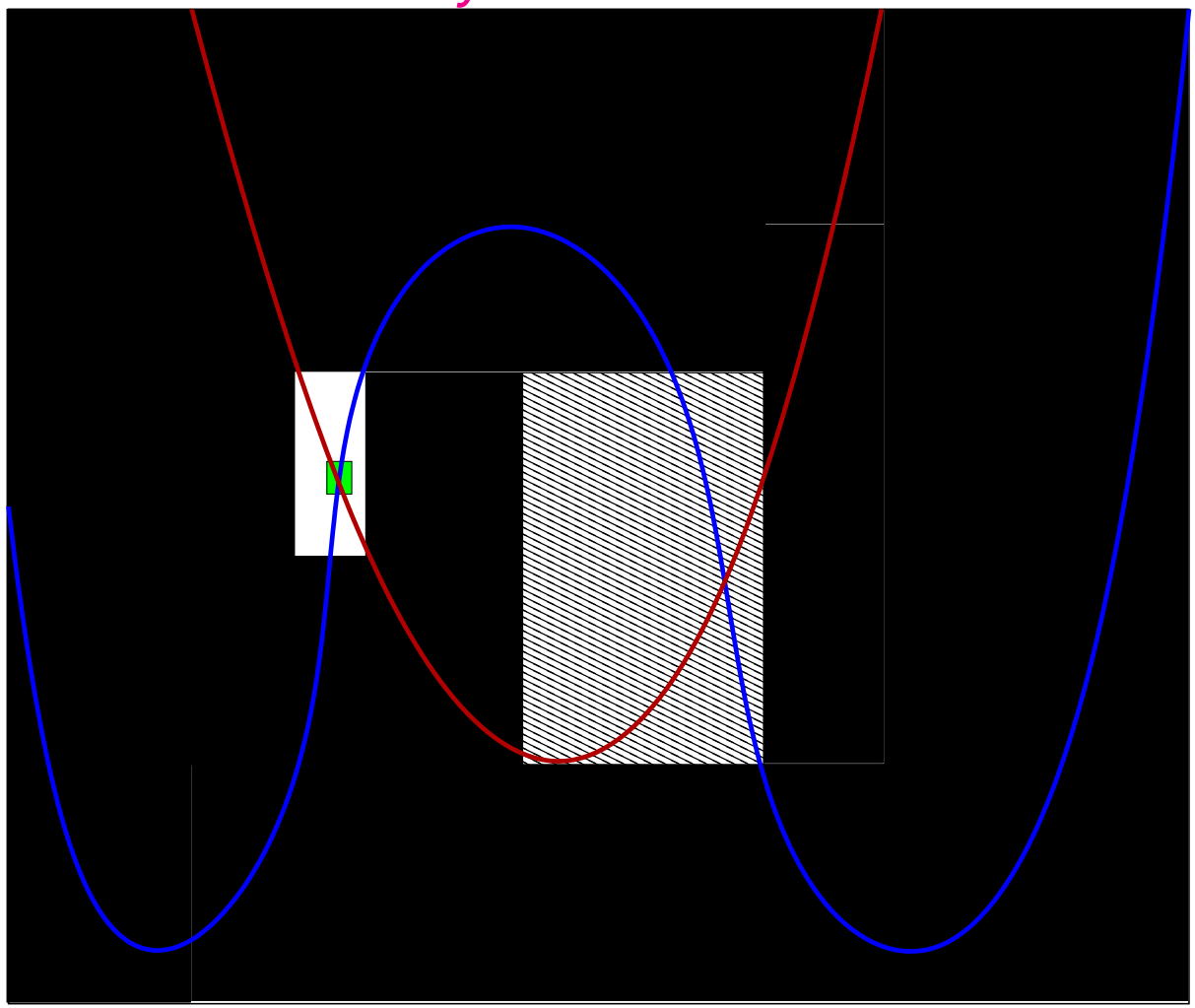
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



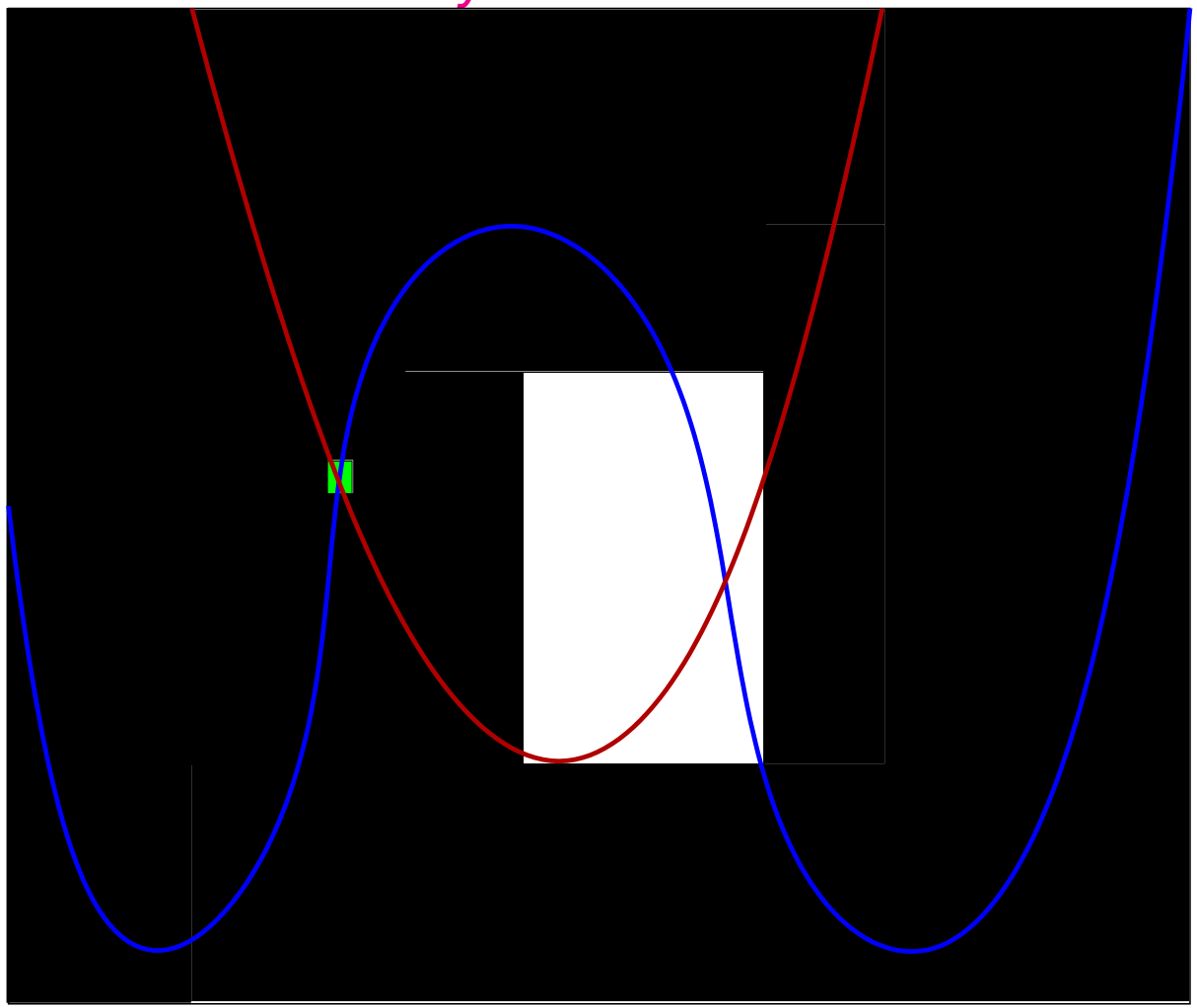
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



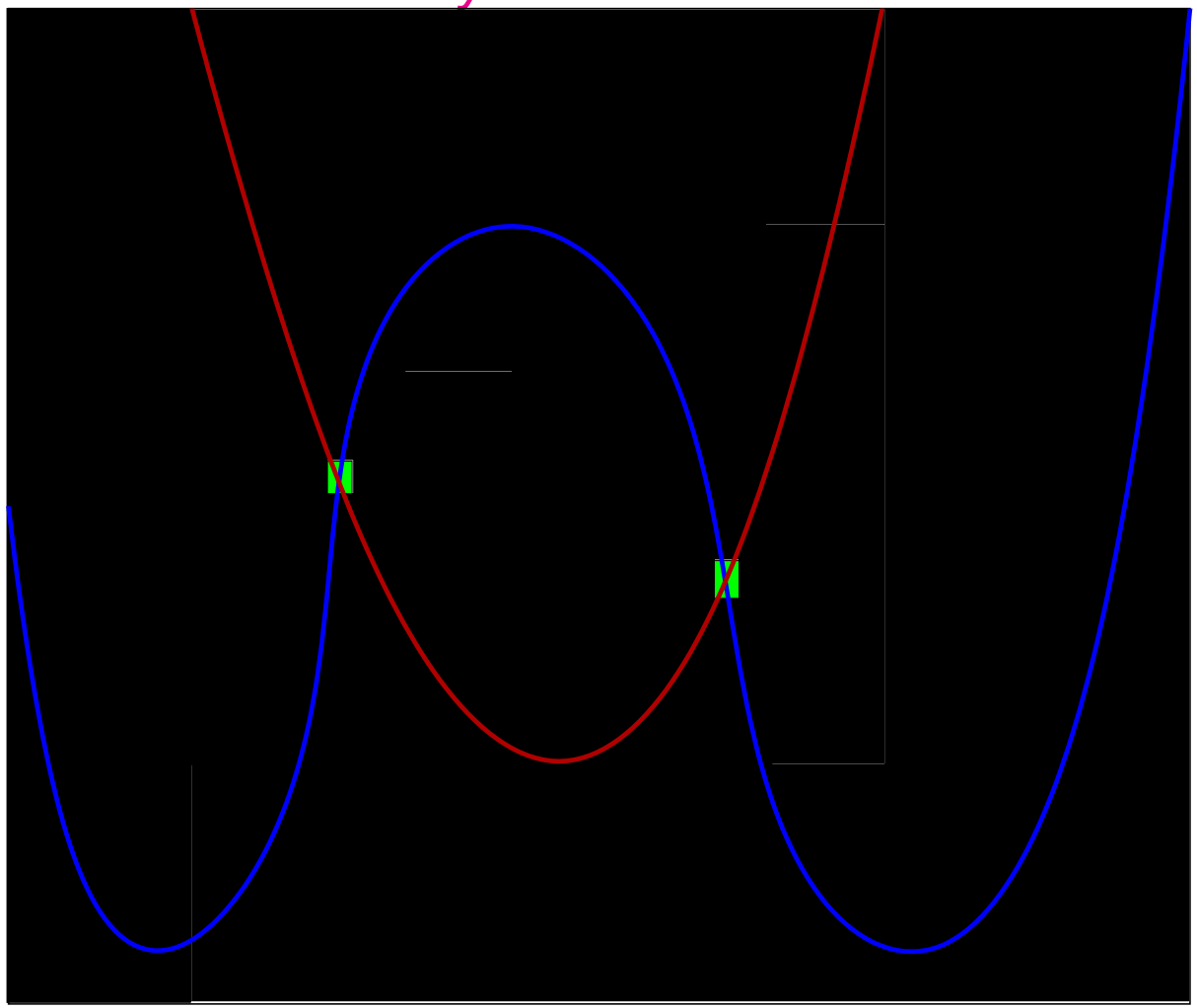
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



CSP over continuous domains : Local consistencies

Working with a single constraint

Consider $D_x = [\underline{x}, \overline{x}]$ and $c(x, x_1, \ldots, x_n)$

If $c(x, x_1, \ldots, x_n)$ does not hold for any values $a \in [\underline{x}, x']$, then $D_x \rightarrow [x', \overline{x}]$

CSP over continuous domains : 2B–consistency

- A constraint c_j is 2B–consistent if for any variable x_i of c_j, the bounds D_{x_i} and $\overline{D_{x_i}}$ have a support in the domains of all other variables of c_j

  $\rightarrow$ Variable x is 2B–consistent for $f(x, x_1, \ldots, x_n) = 0$ if the lower (resp. upper) bound of the domain of x is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$

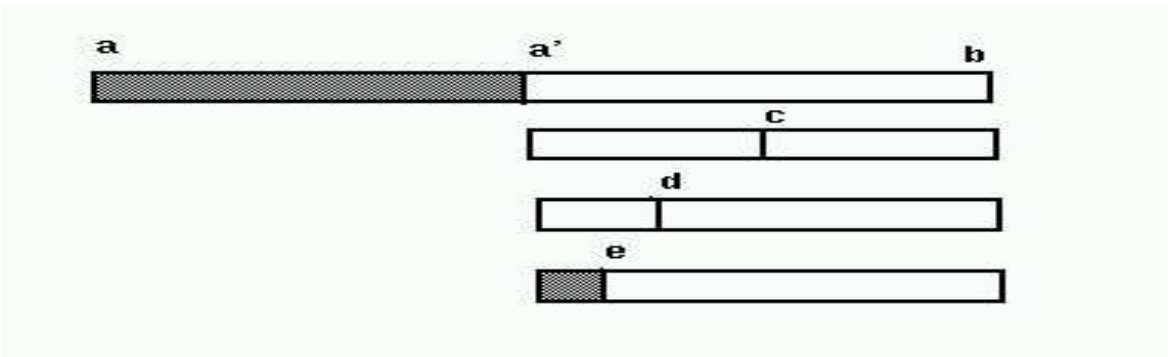
A CSP is 2B–consistent iff all its constraints are 2B–consistent

CSP over continuous domains : 3B–Consistency

3B–Consistency, a shaving process

$\rightarrow$

checks whether 2B–Consistency can be enforced when the domain of a variable is reduced to the value of one of its bounds in the whole system



FPCS: SUM UP

- Correct solver over the floats based on 2B-consistency and 3B-consistency

  no solution are lost

- Projection functions for floats:
  - Direct projections: straightforward adaptation of interval arithmetic
  - Inverse projections: less intuitive, more complex (e.g., might need a larger format than the system variables)
- Handling of rounding modes, nonlinear expressions and the usual mathematical functions (trigonometric. . .)
- Handling of x86 architecture specifics

RAICP: Combining AI and CP

Successive exploration and merging steps

- Use of AI to compute a first approximation of the values of variables at a program node where two branches join

- Building a constraint system for each branch between two join nodes in the CFG of the program and use of CP local consistencies to shrink the domains computed by AI

RAICP: Filtering techniques

- FPCS: 3B(w)-consistency over the floats
  - Projection functions for floats
  - Handling of rounding modes
  - Handling of x86 architecture specifics

- RealPaver: 2B(w)-consistency & Box-consistency over the reals
  - Reliable approximations of continuous solution sets
  - Correctly rounded interval methods and constraint satisfaction techniques

Experiments: eliminating false alarms

CDFL: Program analyzer for proving the absence of runtime errors in program with floating-point computations based on Conflict-Driven Learning

	RAICP	FLUCTUAT	CDFL
False alarms	0	11	0
Total time	40.55 s	18.37 s	208.99 s

Computed on the 55 benchs from CDFL paper (TACAS'12, *D'Silva, Leopold Haller, Daniel Kroening, Michael Tautschnig*)

Computing test-cases: CPBPV_FP

Inputs of CPBPV_FP:

- P, an annotated program
- x, a variable used in a critical test
- $[\underline{x_F}, \overline{x_F}]$, a suspicious interval for x

Annotations of P:

$\rightarrow$ specify the range of the input variables of P and the suspicious interval for x

$\rightarrow$ posted just before a critical test using variable x

Computing test-cases: CPBPV_FP cont.

CPBPV_FP outputs:

- A test case
  $\rightarrow$ *P can produce* a suspicious value for *x*

- A proof that no test case exists
  $\rightarrow$ the suspicious interval *can be removed*

  Only the case if the loops in P cannot be unfolded beyond the bound k

- An inconclusive answer
  $\rightarrow$ *P may produce* a suspicious value

  no test case could be generated
  but the loops in P could be unfolded beyond the bound k

Experiments

Conditions:

- Two cases of floating-point arithmetic pitfalls: programs with *cancellation* and *absorption* phenomena
- C programs handling IEEE 754 FP arithmetic:
  - → compiled with GCC without any optimisation option
  - → run on an x86_64 architecture managed by a 64-bit Linux operating system
- Rounding mode was set to the nearest
- On an Intel Core 2 Duo at 2.8 GHz with 4 GB of memory running 64-bit Linux

Example of program: Heron

→ Uses Heron's formula to compute the area of a triangle from the lengths of a, b, and c (a being the longest side) :

$$\texttt{area} = \sqrt{s * (s-a) * (s-b) * (s-c)}$$

where $\texttt{s} = (a+b+c)/2$.

```
1  /* Pre-condition :  a ≥ b and a ≥ c */
2  float heron(float a, float b, float c) {
3    float s, squared_area;
4
5    squared_area = 0.0f;
6    if (a <= b + c) {
7      s = (a + b + c) / 2.0f;
8      squared_area = s*(s-a)*(s-b)*(s-c);
9    }
10   return sqrt(squared_area);
11 }
```

→ cancellation problems

Experiments: tools

- CDFL: Program analyzer for proving the absence of runtime errors in program with floating-point computations based on Conflict-Driven Learning
- CBMC: state of art bounded mode checkers
- CPBPV_FP: our constraint-based framework

Experiments results

Name	Condition	FPCS	CDFL	CBMC	CPBPV_FP	sol?
`slope`	$res < 26.0_f - 1.0_f$	0.020s	2.014s	1.548s	0.624s	yes
	$res > 26.0_f + 1.0_f$	0.022s	1.599s	0.653s	0.603s	yes
	$res < 26.0_f - 10.0_f$	0.007s	0.715s	1.108s	0.588s	no
	$res > 26.0_f + 10.0_f$	0.007s	1.025s	1.080s	0.593s	no
`heron`	$sq\_aera < 10_f^{-5}$	0.655s	3.874s	0.280s	1.109s	yes
	$sq\_area > 156.25_f + 10_f^{-5}$	1.412s	$>$ 1200s	34.512s	2.294s	yes
`optimised heron`	$sq\_area < 10_f^{-5}$	0.262s	7.618s	0.932s	0.982s	yes
	$sq\_area > 156.25_f + 10_f^{-5}$	37.352s	$>$ 1200s	$>$ 1200s	95.890s	no
`square`	$S > 1.453125$	0.011s	– –	1.079s	0.608s	no

Discussion

- Contribution
  - → Combining AI and CP pays off
  - → Performances of FPCS are encouraging: CP provides an efficient *refutation* framework for FP-constraints

  Details : **Identifying suspicious values in programs with floating-point numbers** *Automated Software Engineering* Journal, May 2014

- Further work
  - → Improving FPCS → hybrid approach to solve constraints over the floating point number
  - → Tighter connexion between CP and AI

Improving FPCS

Limitations

- Based on a local propagation algorithm
- Suffers from multiple occurrence issues,
  e.g. $k * x * x * x$ handled as $k * x * y * z$
- Some "*in house*" solutions: kB consistencies ($\approx$ shaving)

How to improve the solver ?

- Introduce more *global* approach
  (though the strong limitations of floating point arithmetic)
- How ? approximate over R the initial problem over F
  to allow to reduce domain variables using solvers over R