

Robustness analysis of finite precision implementations

Eric Goubault and Sylvie Putot
Cosynus, LIX, Ecole Polytechnique

Journée MEA, calcul ensembliste et interprétation abstraite, 19/03/2015

Context: automatic validation of numerical programs

- Infer invariant properties both in floating-point and real number semantics
 - Abstract interpretation based static analysis (affine arithmetic/zonotopes)
- Bound the implementation errors
- Implemented in the abstract interpreter FLUCTUAT

A difficulty in error analysis: unstable tests

- When finite precision and real control flows are potentially different
- If discontinuity of treatment between branches, joining error analyses from each branch is unsound
- When considering sets of executions, most tests are potentially unstable (just issuing a warning is not practical)

We propose here to compute discontinuity errors in unstable tests

- Bound the difference between the computation in two branches under conditions of unstable tests (mix constraint solving / affine arithmetic)
- Makes our error analysis sound in presence of unstable tests
- Provides a robustness analysis of the implementation

A typical example of unstable tests: affine interpolators

All tests are unstable, but the implementation is robust, the conditional block does not introduce a discontinuity

The screenshot displays the Fluctuat IDE interface. On the left, a C program is shown with line numbers 9 to 42. The program defines a function `main` that takes a float `epsilon` and returns a float `res`. It uses two pairs of 3D vectors, `R1` and `R2`, and performs conditional assignments based on the magnitude of `epsilon` relative to the components of `R1`. A call to `FSENSITIVITY` is made on the result `res`.

In the center, a "Warnings" dialog box is open, titled "Potential overflows:". It lists two threats: "Unstable test (machine and real value do not t...".

On the right, a bar chart shows the results of the analysis. The y-axis represents values from $0.00e+00$ to $1.13e-05$. The x-axis shows indices 0, 25, and 50. A prominent purple bar is visible at index 25, reaching a value of approximately $1.13e-05$.

At the bottom right, a "Variables / Files" panel lists the variables: `R1[2].x`, `R1[2].y`, `R2[0]`, `R2[1]`, `R2[2]`, `main`, and `res`. Below this, a "Variable Interval" table provides numerical ranges for each variable.

Variable	Interval
Float :	0 3.32500000e1
Real :	-8.58306885e-6 3.32500001e1
Global error :	-1.44600869e-5 1.00731850e-5
Relative error :	-∞ ∞
Higher Order error :	0 0
At current point (31) : *	-1.22666e-05 1.22666e-05

At the bottom of the IDE, the status bar indicates: "Last analysis : 0.01 sec / 16384 Kilo Bytes".

But discontinuities also actually occur (sqrt approx.)

The screenshot displays a software interface for analyzing a C program. The main window is titled "Fluctuat - Newnewsqrt".

Code Editor: The code defines a function `sqrt2` with a precision of `1.414213538169860839843750`. It includes `daed_builtins.h` and `math.h`. The `main` function calls `__BUILTIN_DAED_DREAL_WITH_ERROR(1,2,0,0.001)` and then checks `if (x >= 2)`. The code calculates `y` based on the value of `x`.

Warnings: A "Warnings" dialog box is open, showing a "Potential overflows" section and a "Threats" section. The threat is "1 Unstable test (machine and real value do not take)".

Plot: A plot shows a vertical line at `x=2` with a value of `3.59e-02`. The y-axis ranges from `0.00e+00` to `3.59e-02`. The x-axis has markers at `0`, `25`, and `50`.

Variables / Files: The variables are `signgam (integer)`, `x (double)`, and `y (double)`. The file `newnewsqrt.c` is also listed.

Variable Interval: The interval for `x` is `1.00000000` to `1.45362502`. The interval for `y` is `-3.94114776e-2` to `3.89556561e-2`. The global error is `1.00000000` to `1.45312500`. The relative error is `-3.94114776e-2` to `3.89556561e-2`. The higher order error is `0` to `0`. The current point (10) is `-0.0389847` to `0.0389847`.

Last analysis: 0.00 sec / 16384 Kilo Bytes

Fluctuat: computes rounding errors and propagates errors and uncertainties

- Relying on affine forms both for real value and error terms;
- With two sets of constraints on the noise symbols, resp. corresponding to real and finite control flows

Main idea to interpret test, informally

- Affine forms are unchanged, translate the test condition as a condition on the noise symbols ε_i
- The test condition is interpreted as a restriction of the set of inputs, that lead to an execution satisfying the condition: functional interpretation

Example

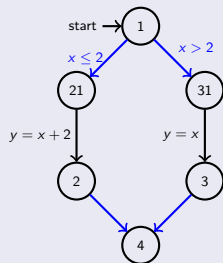
```
real x = [0,10];  
real y = 2*x;  
if (y >= 10)  
  y = x;
```

- Affine forms before tests: $x = 5 + 5\varepsilon_1$, $y = 10 + 10\varepsilon_1$
- In the if branch $\varepsilon_1 \geq 0$: condition acts on both x and y

Example

```

x=[1,3] + u; // [1]
if (x <= 2) // [2-1]
  y = x+2; // [2]
else // [3-1]
  y = x; // [3]
// [4]
}
    
```



$$\begin{aligned}
 (x_1, y_1) &= (2 + \varepsilon_1 + u, \top); & \varepsilon_1 \in \Phi_1^X &= [-1, 1] \\
 (x_{21}, y_{21}) &= (x_1, y_1) \cap (x \leq 2); & \varepsilon_1 \in \Phi_{21}^X &= [0, 1] \\
 (x_2, y_2) &= (x_{21}, x_{21} + 2); & \varepsilon_1 \in \Phi_{21}^X & \\
 (x_{31}, y_{31}) &= (x_1, y_1) \cap (x > 2); & \varepsilon_1 \in \Phi_{31}^X &= [-1, 0] \\
 (x_3, y_3) &= (x_{31}, x_{31}); & \varepsilon_1 \in \Phi_{31}^X & \\
 (x_4, y_4) &= (x_2, y_2) \cup (x_3, y_3); & \varepsilon_1 \in \Phi_4^X &= [-1, 1]
 \end{aligned}$$

Abstract value at each control point c

- For each variable, affine forms for real value and error:

$$f^x = \underbrace{(\alpha_0^x + \bigoplus_i \alpha_i^x \varepsilon_i^r)}_{\text{real value}} + \underbrace{e_0^x}_{\text{center of the error}} + \underbrace{\bigoplus_i \varepsilon_i^x \varepsilon_i^e}_{\text{uncertainty on error due to point } i} + \underbrace{\bigoplus_i m_i^x \varepsilon_i^r}_{\text{propag of uncertainty on value at pt } i}$$

- Constraints on noise symbols coming from interpretation of test condition
 - $\varepsilon^r \in \Phi_r^X$ for real control flow (test on the r^x : constraints on the ε_i^r)
 - $(\varepsilon^r, \varepsilon^e) \in \Phi_f^X$ for finite precision control flow (test on the $f^x = r^x + e^x$: constraints on the ε_i^r and ε_i^e)

Unstable test condition = intersection of constraints $\varepsilon^r \in \Phi_r^X \cap \Phi_f^Y$:

- unstable test: for a same execution (same values of the noise symbols ε_i) the control flow is different
- restricts the range of the ε_i : allows us to bound accurately the discontinuity error

Abstract value

An abstract value X , for a program with p variables x_1, \dots, x_p , is a tuple $X = (R^X, E^X, D^X, \Phi_r^X, \Phi_f^X)$ composed of the following affine sets and constraints, for all $k = 1, \dots, p$:

$$\left\{ \begin{array}{l} R^X : \hat{r}_k^X = r_{0,k}^X + \sum_{i=1}^n r_{i,k}^X \varepsilon_i^r \quad \text{where } \varepsilon^r \in \Phi_r^X \\ E^X : \hat{e}_k^X = e_{0,k}^X + \sum_{i=1}^n e_{i,k}^X \varepsilon_i^r + \sum_{j=1}^m e_{n+j,k}^X \varepsilon_j^e \quad \text{where } (\varepsilon^r, \varepsilon^e) \in \Phi_f^X \\ D^X : \hat{d}_k^X = d_{0,k}^X + \sum_{i=1}^o d_{i,k}^X \varepsilon_i^d \\ \hat{f}_k^X = \hat{r}_k^X + \hat{e}_k^X \quad \text{where } (\varepsilon^r, \varepsilon^e) \in \Phi_f^X \end{array} \right.$$

E^X is the propagated rounding error, D^X the propagated discontinuity error

New discontinuity errors computed when joining branches of a possibly unstable test

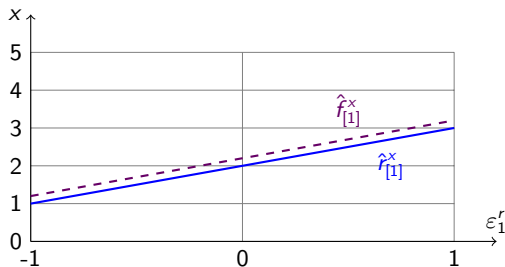
$Z = X \sqcup Y$ is $Z = (R^Z, E^Z, D^Z, \Phi_r^Z, \Phi_f^Z)$ such that

$$\left\{ \begin{array}{l} (R^Z, \Phi_r^Z \cup \Phi_f^Z) = (R^X, \Phi_r^X \cup \Phi_f^X) \sqcup (R^Y, \Phi_r^Y \cup \Phi_f^Y) \\ (E^Z, \Phi_f^Z) = (E^X, \Phi_f^X) \sqcup (E^Y, \Phi_f^Y) \\ D^Z = D^X \sqcup D^Y \sqcup (R^X - R^Y, \Phi_f^X \cap \Phi_f^Y) \sqcup (R^Y - R^X, \Phi_f^Y \cap \Phi_f^X) \end{array} \right.$$


```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

At cpt 1: $\hat{r}_{[1]}^x = 2 + \epsilon_1^r$; $\hat{e}_{[1]}^x = u$



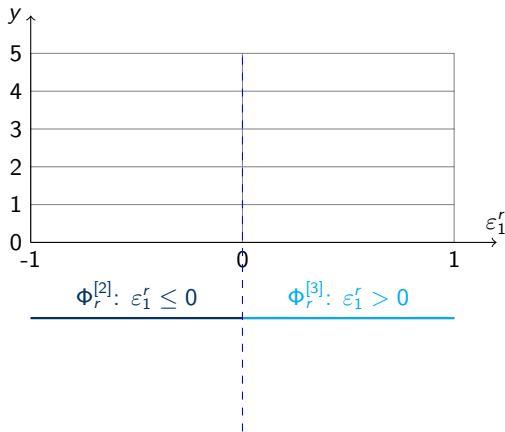
```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

$$\hat{r}_{[1]}^x = 2 + \varepsilon_1^r; \hat{e}_{[1]}^x = u$$

Test $x \leq 2$, real flow:

$$\Phi_r^{[2]} : \hat{r}_{[1]}^x = 2 + \varepsilon_1^r \leq 2$$



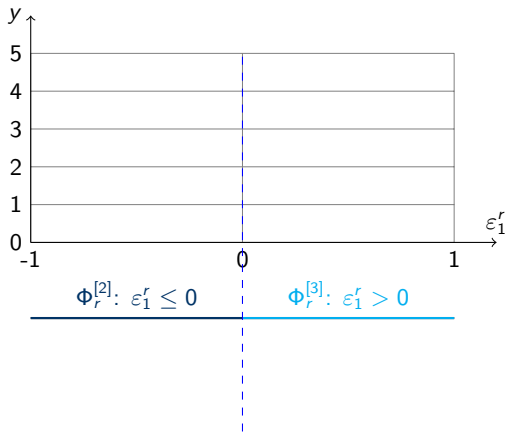
Example: sound unstable test analysis

```
x := [1,3] + u; // [1]
if (x ≤ 2)
  y = x+2; // [2]
else
  y = x; // [3]
// [4]
```

$$\hat{r}_{[1]}^x = 2 + \varepsilon_1^r; \hat{e}_{[1]}^x = u$$

Real at [2]: ($\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]}$)

Real at [3]: ($\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]}$)



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

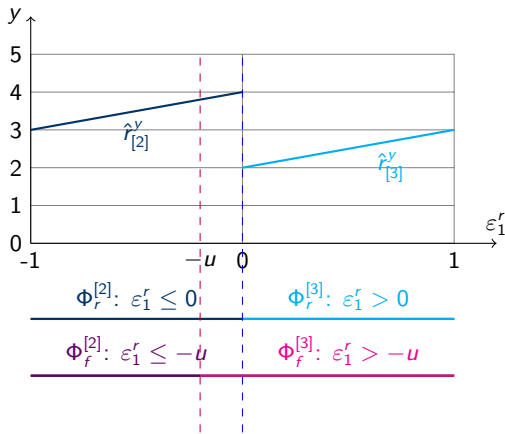
$$\hat{r}_{[1]}^x = 2 + \varepsilon_1^r; \hat{e}_{[1]}^x = u$$

Real at [2]: ($\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]}$)

Real at [3]: ($\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]}$)

Test $x \leq 2$, float flow:

$$\Phi_f^{[2]}: \hat{r}_{[1]}^x + \hat{e}_{[1]}^x = 2 + \varepsilon_1^r + u \leq 2$$



Example: sound unstable test analysis

```

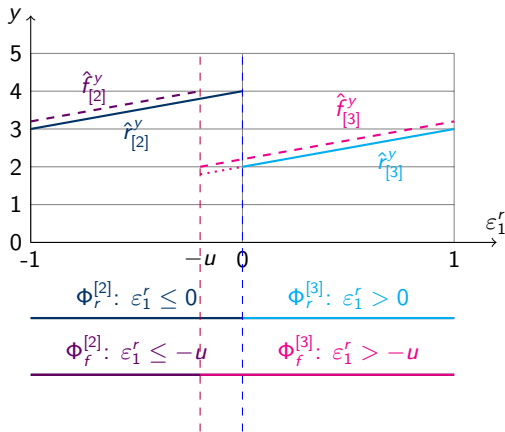
x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

Real at [2]: ($\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]}$)

Real at [3]: ($\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]}$)

Error at [2]: $\hat{e}_{[2]}^y = \hat{e}_{[1]}^x + \delta \varepsilon_2^e$

Error at [3]: $\hat{e}_{[3]}^y = \hat{e}_{[1]}^x$



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

Real at [2]: ($\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]}$)

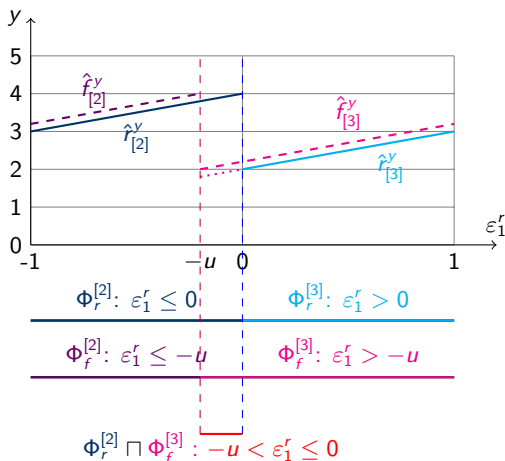
Real at [3]: ($\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]}$)

Error at [2]: $\hat{e}_{[2]}^y = u + \delta\varepsilon_2^e$

Error at [3]: $\hat{e}_{[3]}^y = u$

Unstable test, first possibility:

$\Phi_r^{[2]} \cap \Phi_f^{[3]} : -u < \varepsilon_1^r \leq 0$



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

Real at [2]: ($\hat{r}_{[2]}^y = 4 + \varepsilon_1^r$, $\Phi_r^{[2]}$)

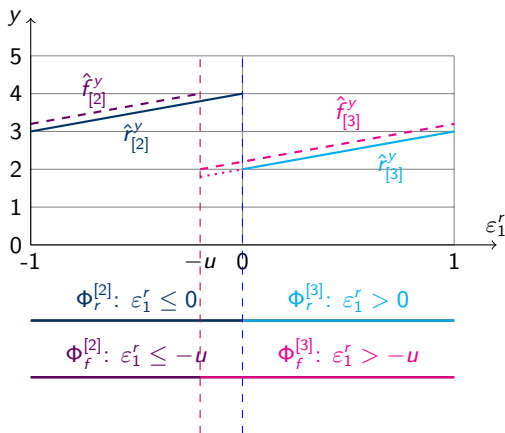
Real at [3]: ($\hat{r}_{[3]}^y = 2 + \varepsilon_1^r$, $\Phi_r^{[3]}$)

Error at [2]: $\hat{e}_{[2]}^y = u + \delta\varepsilon_2^e$

Error at [3]: $\hat{e}_{[3]}^y = u$

Unstable test, second possibility:

$$\Phi_r^{[3]} \cap \Phi_f^{[2]} = \emptyset$$



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

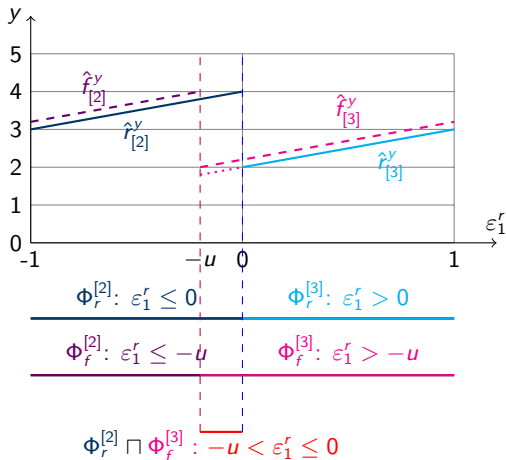
Real at [2]: $(\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]})$

Real at [3]: $(\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]})$

Error at [2]: $\hat{e}_{[2]}^y = u + \delta\varepsilon_2^e$

Error at [3]: $\hat{e}_{[3]}^y = u$

Error at [4]: $\hat{e}_{[2]}^y \sqcup \hat{e}_{[3]}^y \sqcup (\underbrace{\hat{r}_{[3]}^y - \hat{r}_{[2]}^y}_{\hat{r}_{[3]}^y + \hat{e}_{[3]}^y - \hat{r}_{[2]}^y}, \Phi_f^{[3]} \sqcap \Phi_r^{[2]}) = \hat{e}_{[2]}^y \sqcup \hat{e}_{[3]}^y + (\hat{r}_{[3]}^y - \hat{r}_{[2]}^y, \Phi_f^{[3]} \sqcap \Phi_r^{[2]})$



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```

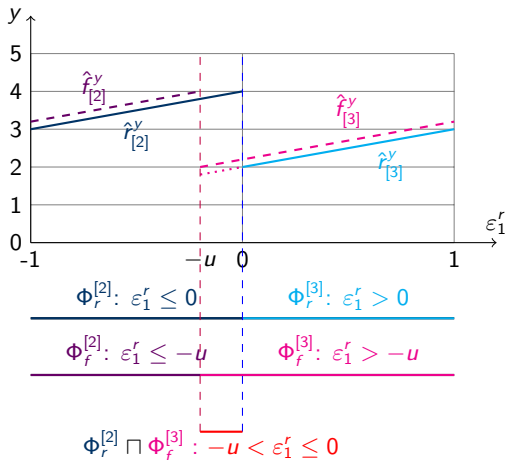
Real at [2]: $(\hat{r}_{[2]}^y = 4 + \varepsilon_1^r, \Phi_r^{[2]})$

Real at [3]: $(\hat{r}_{[3]}^y = 2 + \varepsilon_1^r, \Phi_r^{[3]})$

Error at [2]: $\hat{e}_{[2]}^y = u + \delta\varepsilon_2^e$

Error at [3]: $\hat{e}_{[3]}^y = u$

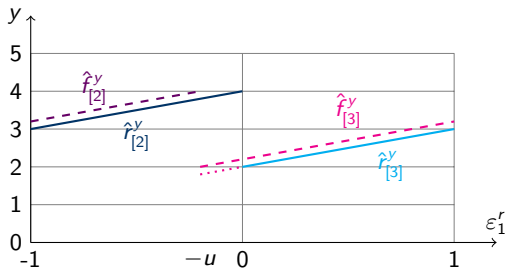
Error at [4]: $\underbrace{\hat{e}_{[2]}^y \sqcup \hat{e}_{[3]}^y}_{\hat{e}_{[4]}^y} + \underbrace{(\hat{r}_{[3]}^y - \hat{r}_{[2]}^y, \Phi_f^{[3]} \cap \Phi_r^{[2]})}_{\hat{d}_{[4]}^y} = u + \delta\varepsilon_2^e - 2\chi_{[-u,0]}(\varepsilon_1^r)$



Example: sound unstable test analysis

```

x := [1,3] + u; // [1]
if (x ≤ 2)
y = x+2; // [2]
else
y = x; // [3]
// [4]
    
```



$$\text{Error at [4]} : \underbrace{\hat{e}_{[2]}^y \sqcup \hat{e}_{[3]}^y}_{\hat{e}_{[4]}^y} + \underbrace{(\hat{r}_{[3]}^y - \hat{r}_{[2]}^y, \Phi_f^{[3]} \sqcap \Phi_r^{[2]})}_{\hat{\partial}_{[4]}^y} = \underbrace{u + \delta \varepsilon_2^e}_{\hat{e}_{[4]}^y} + \underbrace{-2\chi_{[-u,0]}(\varepsilon_1^r)}_{\hat{\partial}_{[4]}^y}$$

$$\text{Real value } \hat{r}_{[4]}^y = 3 + \varepsilon_1^r \in [2, 4]$$

$$\text{Float value } \hat{f}_{[4]}^y = \hat{r}_{[4]}^y + \hat{e}_{[4]}^y = 3 + \varepsilon_1^r + u + \delta \varepsilon_2^e \in [2 + u - \delta, 4 + u + \delta]$$

Householder algorithm for square root

The screenshot displays a software environment with the following components:

- Code Editor:** Contains C code for a Householder algorithm. Line 12, `while (fabs(residu) > EPS) {`, is highlighted in red. The code includes headers, defines `EPS`, and implements a loop to refine the square root of a number.
- Plot:** A graph showing the magnitude of residuals over iterations. The y-axis ranges from $0.00e+00$ to $8.45e-07$. A sharp peak is visible at iteration 17, reaching approximately $8.45e-07$.
- Warnings:** A window titled "Warnings" lists three potential overflows:
 - 1 Error at top in i
 - 1 Value at top in i
- Threats:** A window titled "Threats" lists three items:
 - 1 Unstable test (machine and real value do not take the s...
 - 2 Unstable test (machine and real value do not take the s...
 - 3 BUILTIN bounds not exactly represented
- Variables / Files:** Lists variables: Input (float), Output (float), i (integer), main (integer), residu (float), should_be_zero (float), signgam (integer), and Householder_sqrt.c.
- Variable Interval:** Shows numerical intervals for various variables:
 - Float: $-1.18123876e-6$ to $1.18123956e-6$
 - Real: $-1.02630258e-8$ to $1.02636675e-8$
 - Global error: $-1.17097598e-6$ to $1.17097576e-6$
 - Relative error: -00 to $+00$
 - Higher Order error: 0 to 0
 - At current point (17) : * $-9.17837e-07$ to $9.17837e-07$
- Status Bar:** Shows "Last analysis : 0.42 sec / 28672 Kilo Bytes".

- Error bounds now sound even with unstable tests
- Candidate input values for unstable tests given by $\Phi_f^X \sqcap \Phi_r^Y$
- Robustness analysis and discontinuity error bounds applicable to general uncertainties (not only finite precision)
- A.I. techniques allow to extend error analyses compared to more classical interval-like techniques
- Natural potential on these robustness aspects for more interaction with constraint-based approaches to the verification of finite-precision implementations such as, e. g. O. Ponsini, C. Michel, M. Rueher: Refining Abstract Interpretation Based Value Analysis with Constraint Programming Techniques. CP 2012