Rigorous enclosure of round-off errors in floating-point computations

Rémy Garcia  Claude Michel  Michel Rueher

Université Côte d’Azur, CNRS, I3S, France
Outline

Motivation

Floating-point numbers

A constraint system to bound round-off errors

Rigorous enclosure of round-off errors

Experimentation

Conclusion
Motivation

- Program on $\mathbb{F}$ written with the semantic of $\mathbb{R}$
  - $\mathbb{F} \neq \mathbb{R}$
  - Computation over $\mathbb{F}$ produce errors

- Error analysis tools (Fluctuat, FPTaylor, PRECiSA, ...)
  compute an **over-approximation** of the error
  $\rightarrow$ **Computed bounds** of error are rarely **reachable**

- Other tools (S3FP, FPSDP, ...) compute an **under-approximation** of the largest absolute error
  $\rightarrow$ Possible **over-approximation** of bounds

None of these tools provides an **enclosure** of the **largest absolute error**
Motivating example

Consider the following program that compute $z$ and use a conditional to raise an alarm or proceed without it:

$$z = \frac{(3x+y)}{w};$$

```java
if (z - 10 <= δ) {
    proceed();
} else {
    raiseAlarm();
}
```

Critical issue

Is the error on $z$ small enough to avoid raising the alarm when the value of $z$ is less than or equal to 10 on $\mathbb{R}$?
Motivating example (cont.)

Computation is done over 64-bit floating-point numbers with $x \in [7, 9]$, $y \in [3, 5]$, $w \in [2, 4]$, and $\delta$ set to $5.32e-15$.

<table>
<thead>
<tr>
<th></th>
<th>FPTaylor</th>
<th>PRECiSA</th>
<th>Fluctuat</th>
<th>FErA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{e}_z$</td>
<td>5.15e-15</td>
<td>5.08e-15</td>
<td>6.28e-15</td>
<td>4.96e-15</td>
</tr>
</tbody>
</table>

Bounds are smaller than $\delta$ $\rightarrow$ no alarm for $z \leq 10$.

FErA output an enclosure of $[3.55e-15, 4.96e-15]$ for $e_z$ with

- $x = 8.99999999999996624922$ $e_x = -8.88178419700125232339e-16$
- $y = 4.99999999999994848565$ $e_y = -4.44089209850062616169e-16$
- $w = 3.19999999999998419042$ $e_w = +2.22044604925031308085e-16$
- $z = 10.00000000000000035527$ $e_z = -3.55271367880050092936e-15$
Let us change \( \delta \) to \( 3.55e-15 \)

FErA provides one case where the else branch is taken and \textbf{input values} exercising it

\[
\begin{align*}
  x &= 8.9999999999996624922 & e_x &= -8.88178419700125232339e-16 \\
  y &= 4.9999999999994848565 & e_y &= -4.44089209850062616169e-16 \\
  w &= 3.1999999999998419042 & e_w &= +2.22044604925031308085e-16 \\
  z &= 10.0000000000000035527 & e_z &= -3.55271367880050092936e-15
\end{align*}
\]

Fluctuat, FTPaylor, and PRECiSA are unable to do so, as they only compute an \textbf{over-approximation} of errors
Floats – definition

- $F$ is a finite subset of $\mathbb{R}$

$-\infty$ \hspace{1cm} $0$ \hspace{1cm} $+\infty$

$\mathbb{R}$ (horizontal line) and $F$ (vertical lines)

- IEEE 754 floats are represented by
  - a sign
  - a mantissa
  - an exponent

$(-1)^s \times 1.m \times 2^e$

$(-1)^0 \times 1.00010001101011010 \ldots 0 \times 2^{-1} \approx 0.53452301025390625$
Floats – rounding

Problem: \( x, y \in \mathbb{F} \rightarrow x \cdot y \notin \mathbb{F} \), where \( \cdot \) is an operation on \( \mathbb{R} \)

- **Require rounding** \( \circ \) of the result to the closest float
  - Loss of **precision** \( \circ(x) \) usually not equal to \( x \)
  - Root of the **divergence** between \( \mathbb{R} \) and \( \mathbb{F} \)

\[
\circ(0.1) = 0.100000000001490116119384765625
\]

- **Rounding accumulation**
  - Rounding on **all operations** \( \circ(\circ(x \cdot y) \cdot z) \)
  - Increase the **divergence** between \( \mathbb{R} \) and \( \mathbb{F} \)
  - Reduce or cancel an error with **error compensation**

\[
\circ(\circ(0.1 + 0.2) + 0.2) = 0.5
\]

\[
e = -e \approx 7.4505805969238281e-09
\]
Constraint Programming (CP) is a paradigm used for solving NP-complete combinatorial problems.

Explicit separation between
- **modelling**, which is a formalisation of the problem,
- and **solving**, that uses dedicated techniques to find a solution.
How does CP work?

Modelling
A CSP \((\mathcal{X}, \mathcal{D}, \mathcal{C})\) is defined by:

- \(\mathcal{X}\) is the set of variables
- \(\mathcal{D}\) is the set of domains
  - A domain is the set of all possible values for each \(x \in \mathcal{X}\)
- \(\mathcal{C}\) is the set of constraints
  - A constraint is a relation between variables

Solving
- **Filtering**, removes trivially inconsistent values from domains of variables in a constraint
  - **Propagate** to other constraints with common variables
- **Search**, selects a variable and splits its domain, according to a search strategy

→ The solving process is repeated until a solution is found or when the search space is fully explored
Let $x$ be a floating-point variable of a CSP.

### Domain of values $D_x$
- Interval of $\mathbb{F}$
- **Cannot** represent the associated error ($\notin \mathbb{F}$)

### Domain of errors $D_{ex}$
- Interval of $\mathbb{Q}$
- **Correctly** represents an error
  - For $+, -, \times, \div$
Filtering – error computation

- Compute errors over $\mathbb{Q}$
  - **Exact computation** of errors
    
    $$ e = (x_R \cdot y_R) - (x_F \odot y_F) $$

    with $\cdot$ an operation over $\mathbb{R}$ and $\odot$ an operation over $\mathbb{F}$
    (operations are restricted to $+, -, \times, \div$)

- Signed errors
  - Possible *compensation* of errors
Filtering – domains of variables

**Domain of values** $D_x$

Projection functions from [Michel02], [BotellaGM06], and [MarreM10]

**Domain of errors** $D_{ex}$

Projection functions based on $(x_R \cdot y_R) - (x_F \oplus y_F)$

Error filtering on which constraints?

- **Arithmetic constraints**: $+, -, \times, \div$
- **Assignement constraint**: propagation of the error

Example for $z = x - y$

- $e_z \leftarrow e_z \cap (e_x - e_y + e_\Theta)$
- $e_x \leftarrow e_x \cap (e_z + e_y - e_\Theta)$
- $e_y \leftarrow e_y \cap (e_x - e_z + e_\Theta)$
- $e_\Theta \leftarrow e_\Theta \cap (e_z - e_x + e_y)$

A constraint system to bound round-off errors
Consider $z = x \odot y$

IEEE 754, operations correctly rounded: $\oplus, \ominus, \otimes, \oslash$

$$(x \odot y) - \frac{1}{2} \text{ulp}(x \odot y) \leq x \cdot y \leq (x \odot y) + \frac{1}{2} \text{ulp}(x \odot y)$$

ulp: distance between two consecutive floats

Error on the operation

$$-\frac{1}{2} \text{ulp}(x \odot y) \leq e_\odot \leq +\frac{1}{2} \text{ulp}(x \odot y)$$
New notation for constraints over errors

$$\text{err}(x) \geq \epsilon$$

- $\text{err}(x)$ represent the domain of errors of variable $x$
- $\text{err}(x) \in \mathbb{Q}$, the constraint is over $\mathbb{Q}$

Modelize a program as an optimization problem

$$\max | \text{err}(x) |$$
Branch-and-bound – schema

- Computes two bounds of errors:
  - Dual: **upper bound** of error, $\bar{e}$ (filtering)
  - over-approximation
  - Primal: **lower bound** of error, $\hat{e}$ (generate-and-test)
  - reachable $\rightarrow$ provides input values

- Error **maximization** directed by **search on values**
  - explore finite search space in $\mathbb{F}$ $\rightarrow$ infer error

- A maximal error is in general **hard** to find

- **Anytime algorithm** $\rightarrow$ provides input values, $e^*$, and $\bar{e}$
Stopping criteria

Operation error: $e_{\circ}$ and $z$ result of operation

\[ e_{\circ} \leq \frac{1}{2}\text{ulp}(z) \]

→ highly dependent on the distribution of floats

Consider interval $(2^n, 2^{n+1})$

- Distance between two floats is the same
- All floats have the same ulp

→ cannot improve $e_{\circ}$ by means of projection functions

Once results for all operations satisfy this criteria, stop the exploration of this part of the search space
A box $B$ can be in one of the following three states:

- **unexplored**

- **discarded**, s.t. $\bar{e}^B \leq e^*$

- **sidelined**, s.t. stopping criteria is true
  - $e^S$ is max $\bar{e}^B$ of sidelined boxes
Bounding – dual computation

Computation based on constraint programming filtering

- projection functions

\[
\begin{align*}
    e_z &\leftarrow e_z \cap (e_x - e_y + e_\Theta) \\
    e_x &\leftarrow e_x \cap (e_z + e_y - e_\Theta) \\
    e_y &\leftarrow e_y \cap (e_x - e_z + e_\Theta) \\
    e_\Theta &\leftarrow e_\Theta \cap (e_z - e_x + e_y)
\end{align*}
\]

For a box \( B \)

- Propagate constraints to filter domains
- Update \( \overline{e} \) with max between:
  - \( \overline{e} \) of unexplored boxes
  - \( \overline{e}^S \) of sidelined boxes
Bounding – primal computation

Generate-and-test: random instantiation of input variables

For each box $B$ repeat $n$ times

- Randomly instantiate input variables with respect to domains of values
- Compute $f_Q() - f_F()$
- Local search ($m$ steps):
  - explore floats around input values
    - guided by the best local value of the error
  - Compute $f_Q() - f_F()$

→ Update $e^*$ with best computed error
Branching – explore boxes

Variable selection

- Choose in **round-robin** order a variable $x$ that is not a singleton

Domain splitting

- Apply a bisection on the domain of values of $x$ to generate two subboxes

Box selection

- Use **best-first search** to select a box $B$ with the greatest upper bound of error
Benchmarks are taken from **FPBench** (see paper)

- (operations are restricted to $+, -, \times, \div$)

FErA over-approximation bound

- Best **twice**
- Second **6 times**
- **Never** the worst

FErA solving time is **reasonable** for most of benchmarks

- only one bench timeout at 10 minutes
Contribution

Rigorous enclosure of round-off errors

- **Enclosure** of a largest absolute error
- Reachable primal $\rightarrow$ provide inputs values **exercising** the error
- Provides a **tighter** $\bar{e}$ $\rightarrow$ removes some **false positives**
Further work

- **Tighter representation** of round-off errors on elementary operations
- Experimentations with different **search strategies**
- More efficient **local search** to speed up the primal computation procedure