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

NSV 2020

Rémy Garcia Claude Michel Michel Rueher

Université Côte d'Azur, CNRS, I3S, France

July 20, 2020

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 = (3*x+y)/w;

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 δ set to $5.32e{-15}$

	FPTaylor	PRECiSA	Fluctuat	**FErA**
$\overline{e}_z$	5.15e-15	5.08e-15	6.28e-15	4.96e-15

Bounds are **smaller** than δ → **no alarm** for $z \leqslant 10$

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

$$x = 8.99999999999996624922 \quad e_x = -8.88178419700125232339e{-16}$$
$$y = 4.99999999999994848565 \quad e_y = -4.44089209850062616169e{-16}$$
$$w = 3.19999999999998419042 \quad e_w = +2.22044604925031308085e{-16}$$
$$z = 10.00000000000000035527 \quad e_z = -3.55271367880050092936e{-15}$$

Motivating example (cont.)

Let us change δ to $3.55e{-}15$

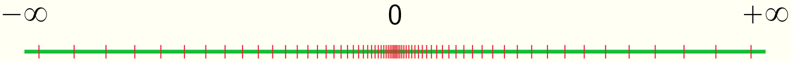
FErA provides one case where the else branch is taken and **input values** exercising it

$$
\begin{aligned}
x &= 8.99999999999996624922 & e_x &= -8.88178419700125232339e{-}16\\
y &= 4.99999999999994848565 & e_y &= -4.44089209850062616169e{-}16\\
w &= 3.19999999999998419042 & e_w &= +2.22044604925031308085e{-}16\\
z &= 10.00000000000000035527 & e_z &= -3.55271367880050092936e{-}15
\end{aligned}
$$

Fluctuat, FPTaylor, and PRECiSA are unable to do so, as they only compute an **over-approximation** of errors

Floats – definition

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



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

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

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

$$(-1)^0 \times 1.0001000110101101 0..0 \times 2^{-1} \approx 0.53452301025390625$$

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 $\mathbb{F}$

$$\circ(0.1) = 0.10000000\mathbf{1490116119384765625}$$

- **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: overview

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 works?

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** inconsistant 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**

$\rightarrow$ The solving process is repeated until a solution is found or when the search space is fully explored

Domain of errors over $\mathbb{Q}$

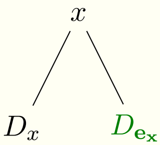
Let x be a floating-point variable of a CSP

Domain of values D_x

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

Domain of errors $D_{\mathbf{ex}}$

- Interval of $\mathbb{Q}$
- **Correctly** represents an error
  - For $+, -, \times, \div$



Filtering – error computation

- Compute errors over $\mathbb{Q}$
  - **Exact computation** of errors

$$e = (\mathbf{x}_{\mathbb{R}} \cdot \mathbf{y}_{\mathbb{R}}) - (\mathbf{x}_{\mathbb{F}} \odot \mathbf{y}_{\mathbb{F}})$$

  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 $D_{\mathbf{e_x}}$

Projection functions based on $(\mathbf{x}_{\mathbb{R}} \cdot \mathbf{y}_{\mathbb{R}}) - (\mathbf{x}_{\mathbb{F}} \odot \mathbf{y}_{\mathbb{F}})$

Error filtering on which constraints?

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

Example for $z = x - y$

$$\mathbf{e_z} \leftarrow \mathbf{e_z} \cap (\mathbf{e_x} - \mathbf{e_y} + \mathbf{e}_{\ominus})$$
$$\mathbf{e_x} \leftarrow \mathbf{e_x} \cap (\mathbf{e_z} + \mathbf{e_y} - \mathbf{e}_{\ominus})$$
$$\mathbf{e_y} \leftarrow \mathbf{e_y} \cap (\mathbf{e_x} - \mathbf{e_z} + \mathbf{e}_{\ominus})$$
$$\mathbf{e}_{\ominus} \leftarrow \mathbf{e}_{\ominus} \cap (\mathbf{e_z} - \mathbf{e_x} + \mathbf{e_y})$$

Filtering – operation error

Consider $z = x \odot y$

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

$$(\mathbf{x} \odot \mathbf{y}) - \frac{1}{2}\,\mathrm{ulp}(\mathbf{x} \odot \mathbf{y}) \leqslant \mathbf{x} \cdot \mathbf{y} \leqslant (\mathbf{x} \odot \mathbf{y}) + \frac{1}{2}\,\mathrm{ulp}(\mathbf{x} \odot \mathbf{y})$$

ulp: distance between two consecutive floats

Error on the operation

$$-\frac{1}{2}\,\mathrm{ulp}(\mathbf{x} \odot \mathbf{y}) \leqslant \mathbf{e}_{\odot} \leqslant +\frac{1}{2}\,\mathrm{ulp}(\mathbf{x} \odot \mathbf{y})$$

Constraint over errors

- **New notation** for constraints over errors

$$err(x) \geqslant \epsilon$$

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

Modelize a program as an optimization problem

$$\max \mid err(x) \mid$$

Branch-and-bound – schema

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

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

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

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

Stopping criteria

Operation error: $e_\odot$ and z result of operation

$$\mathbf{e}_\odot \leqslant \frac{1}{2}\mathsf{ulp}(z)$$

→ 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 $e_\odot$ by means of projection functions

Once results for all operations satisfy this criteria, stop the exploration of this part of the search space

Branch-and-bound – boxes

A box B can be in one of the following three states:

- *unexplored*
- *discarded*, s.t. $\overline{e}^B \leqslant e^*$
- *sidelined*, s.t. stopping criteria is true
  - e^S is max $\overline{e}^B$ of sidelined boxes

Bounding – dual computation

Computation based on constraint programming **filtering**

- projection functions

$$\mathbf{e_z} \leftarrow \mathbf{e_z} \cap (\mathbf{e_x} - \mathbf{e_y} + \mathbf{e_\ominus})$$
$$\mathbf{e_x} \leftarrow \mathbf{e_x} \cap (\mathbf{e_z} + \mathbf{e_y} - \mathbf{e_\ominus})$$
$$\mathbf{e_y} \leftarrow \mathbf{e_y} \cap (\mathbf{e_x} - \mathbf{e_z} + \mathbf{e_\ominus})$$
$$\mathbf{e_\ominus} \leftarrow \mathbf{e_\ominus} \cap (\mathbf{e_z} - \mathbf{e_x} + \mathbf{e_y})$$

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 $f_{\mathbb{Q}}() - f_{\mathbb{F}}()$
- Local search (m steps):
  - explore floats around input values
    - guided by the best local value of the error
  - Compute $f_{\mathbb{Q}}() - f_{\mathbb{F}}()$

→ 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

Experimentation – FPBench

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