A Reduced Product of Absolute and Relative Error Bounds for Floating-point Analysis

Maxime Jacquemin[1], Sylvie Putot[2], and Franck Védrine[1]

[1] CEA, List, Software Reliability and Security Laboratory, PC 174
91191 Gif-Sur-Yvette France
firstname.lastname@cea.fr
[2] LIX, CNRS and École Polytechnique, Palaiseau, France.
putot@lix.polytechnique.fr

Abstract. Rigorous estimation of bounds on errors in finite precision computation has become a key point of many formal verification tools. The primary interest of the use of such tools is generally to obtain worst-case bounds on the absolute errors. However, the natural bound on the elementary error committed by each floating-point arithmetic operation is a bound on the relative error, which suggests that relative error bounds could also play a role in the process of computing tight error estimations. In this work, we introduce a very simple interval-based abstraction, combining absolute and relative error propagations. We demonstrate with a prototype implementation how this simple product allows us in many cases to improve absolute error bounds, and even to often favorably compare with state-of-the art tools, that rely on much more costly relational abstractions or optimization-based estimations.

1 Introduction

Computing worst-case bounds on the potential loss of accuracy in numerical programs due to the use of floating-point arithmetic is of utmost importance in many fields of application, such as embedded systems or numerical simulation. Several analyzes for the computation of sound error bounds have been proposed in the last 15 years, and generally implemented in academic prototypes. Most of them rely on abstractions of the value and absolute errors of program variables. An additional output of such analyses is sometimes bounds on the relative errors, but they are mostly computed a posteriori, from the values and absolute errors. Still, the natural bound on the elementary error committed by each floating-point arithmetic operation is a bound on the relative error. This strongly suggests that relative error bounds can also play a role in the process of computing tight error estimates. This is what this work proposes to explore. We indeed note that on some patterns, abstraction relying only on absolute error yields unreasonably conservative error bounds, and that a simple

product with relative error bounds can bring a drastic improvement. One such pattern is a conditional statement that tests a quantity subject to a rounding error. We consider the very simple piece of code introduced in Example 1.

Example 1. Variables `i, x, y` are double precision floating-point numbers, and input `i` is given without error in range $[1, 100]$: The multiplication `x:=i*i` results in variable `x` in [1,10000] with an elementary absolute rounding error $\mathcal{E}_a(x)$ bounded in $[-9.09e^{-13}, 9.09e^{-13}]$. If evaluated directly by using the fact that the elementary error in floating-point arithmetic is bounded in relative error, we obtain a relative error $\mathcal{E}_r(x)$ bounded in $[-1.11e^{-16}, 1.11e^{-16}]$. It is clear that some information is lost if the error is abstracted by bounds on the absolute error only, especially on a non-relational abstraction like intervals (and the wider the intervals, the more so). Take for instance constraint `x <= 2.0` on our variable `x`. Using the relative error bound allows us to compute a much tighter absolute error bound in the `true` branch of the conditional. Indeed, the value of `x` knowing that the constraint is satisfied can be reduced in [1,2]. Thus, a new absolute error bound for `x` in this branch can be computed by $\mathcal{E}_a(x) = \mathcal{E}_a(x) \cap x\mathcal{E}_r(x) = [-9.09e^{-13}, 9.09e^{-13}] \cap [1, 2][-1.11e^{-16}, 1.11e^{-16}] = [-2.22e^{-16}, 2.22e^{-16}]$. Therefore, the absolute error on variable `y` will be bounded in $[-2.22e^{-16}, 2.22e^{-16}]$. Whereas if not using this reduced product, the error on `x` simply propagates as an error on `y`, and the absolute error bound on `y` will be $[-9.09e^{-13}, 9.09e^{-13}]$.

```
x = i*i;
if (x <= 2.0)
  y = x;
else
  y = 2.0;
```

Another simple example, that focuses on arithmetic operations, is taken from the introduction of [16]:

Example 2. We consider expression $t/(t+1)$, where t is a double precision floating-point value in [0,999]. An error is committed when computing $t + 1$: the absolute error of $t + 1$ is bounded by $\mathcal{E}_a(t + 1) = [-5.68e^{-14}, 5.68e^{-14}]$, the relative error by $\mathcal{E}_r(t+1) = [-1.1e^{-16}, 1.1e^{-16}]$. For comparison, the a posteriori evaluation of the relative error bounds from the absolute error bounds is

$$\frac{\mathcal{E}_a(t+1)}{t+1} = \frac{[-5.68e^{-14}, 5.68e^{-14}]}{[1, 1000]} = [-5.68e^{-14}, 5.68e^{-14}]$$

thus 500 times larger than the direct estimate $\mathcal{E}_r(t + 1)$. Thus, if the relative error is not explicitely propagated, some information is lost.

And indeed, as we will develop in Section 3.2, it is natural to express the absolute error on the division $x \div y$ using the relative error on y. Actually, the bounds on the absolute error of $t/(t + 1)$ using this product are $[-1.67e^{-13}, 1.67e^{-13}]$, 340 times tighter compared to the bounds $[-5.68e^{-11}, 5.68e^{-11}]$ that would be obtained by classical error propagation relying only on absolute error. On this example, the method of [16], that relies on optimisation of the error globally on subexpressions, is more accurate than our improved bounds. This is because we

still suffer here from the conservativeness of interval abstraction in the evaluation of our expressions. But their results come at the expense of much more expensive computations.

In both cases, we note that this conjunction of the propagation of the relative error and the absolute error, in the end, helps us improve sometimes dramatically the absolute error bound, while maintaining a very cheap analysis. It is indeed the center idea of this work to observe that the information contained in the absolute and the relative error bounds are complementary, and to propose an interval-based analysis computing an inexpensive reduced product that combines the information for the best final estimations of error bounds. The idea has been experimented here on a reduced set of operators, and ignoring the possibility of control flow divergences between the floating-point and the corresponding real computations, as a proof of concept. But the approach can naturally be extended to more operators, as well as to relational abstractions of values and error, in order to enhance many existing error analyzes. Additionally, using relational abstractions is necessary to handle with reasonable accuracy errors due to control flow divergences.

Contents After some background on floating-point arithmetic in Section 2, we introduce our abstraction in Section 3. In Section 4, we demonstrate that our analysis, implemented in the Frama-C platform, while being very efficient in time, often also favorably compares in accuracy to the generally much more expensive existing appproaches of the state of the art [16, 11]. We use for this a set of benchmarks classically used to compare error analyses, extracted from FPBench[3].

Related Work Abstract interpretation [2] is widely used for the analysis of floating-point computations. Most analyses dedicated to the propagation of error bounds in floating-point computations focus on absolute roundoff error bounds. Existing abstraction for rounding errors often are based on intervals [13, 8], affine forms [1, 9, 10, 4] as implemented in the analyzer Fluctuat [7]. The tool Gappa [6] relies on interval arithmetic and expression rewriting. It additionally generates a proof of the bounds it computes, that can be automatically checked with a proof assistant such as Coq. Some approaches combine these abstractions with some optimization techniques to enhance bounds on values and errors. The tool PRECiSA [17], relies on intervals, combined with branch-and-bound optimization and symbolic error computations using the Bernstein basis. It also generates proof certificates on the error bounds. Rosa [5] combines affine arithmetic with some SMT solving. Real2float [12] also bounds absolute rounding errors using optimization techniques relying on semidefinite programming and sparse sums of squares certificates.

Some of the tools based on these methods provide the user with relative error bounds, but they are often a posteriori bounds, computed from the bounds on the absolute error. Direct relative error bounds are computed by FPTaylor [16],

[3] http://fpbench.org

which formulates the problem of bounding errors as an optimization problem, using first-order Taylor approximations of arithmetic expressions. The optimization based approach of FPTaylor has been extended in Daisy [11], which also relies on some of the techniques already present in Rosa [5] for value and absolute error estimate. In the present work, we propose a much less costly alternative to the direct estimate of relative error, which we show still behaves very well on a number of classical benchmarks, and demonstrate how this error can be used to also improve absolute error bounds. Compared to the related work which uses optimization somewhat blindly, we demonstrate the interplay between the two types of errors.

2 Floating-point Arithmetic and Rounding Errors

2.1 Floating-point numbers and rounding errors.

The floating-point representation of a real number x is defined by the IEEE 754 standard as the triple (sgn, sig, exp). In this triple, sgn corresponds to the sign of x, the significand sig has fixed size p, and, for normalized numbers, is such that $1 \leq sig < 2$, and exp is the exponent. This representation is evaluated as $(-1)^{sgn} \times sig \times 2^{exp}$. Denormalized numbers allow gradual underflow to zero. Their exponent is fixed equal to $e_{\min}$, and the significand is such that $sig < 1$.

Because of the finite size of the significand, a real value is represented by a rounded value. This rounding can be represented through the operator $\mathrm{rnd} : \mathbb{R} \to \mathbb{F}$ that returns the closest floating-point number with respect to the rounding mode. Common rounding modes defined by the standard are rounding to nearest (ties to even), toward zero and toward $\pm\infty$. In this work, we consider the classical case of rounding to nearest. The rounding operator is often modeled as:

$$\mathrm{rnd}(x) = x(1 + e_x) + d_x \tag{1}$$

where $|e_x| \leq \epsilon_M$, $|d_x| \leq \delta_M$, $e_x \times d_x = 0$ and (ϵ_M, δ_M) are parameters fixed by the format (simple, double or quad precision). Constant ϵ_M is often called the *machine epsilon* and depends of the precision p of the floating-point numbers used. It is equal to the distance 2^{1-p} between 1 and its floating-point successor, with $p = 24$ for float and $p = 53$ for double numbers. Constant δ_M is the smallest denormalized number, equal to $2^{e_{\min}+1-p}$, with $e_{\min} = -127$ for float and $e_{\min} = -1023$ for double numbers. In this model, d_x represents the absolute error committed when rounding to a denormalized floating-point number while e_x is the relative error committed when rounding to a normalized floating-point number. They cannot be present at the same time, which is expressed by condition $e_x \times d_x = 0$.

This model can be refined. The normalized floating-point rounding error xe_x in (1) is actually bounded by the distance between two consecutive floating-point numbers around x. This distance can be expressed as $\mathrm{ufp}(x)\,\epsilon_M$, using the notion

of *unit in the first place* ufp() introduced in [15] and defined by:

$$\mathrm{ufp}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 2^{\lfloor log_2 |x| \rfloor} & \text{if } x \neq 0 \end{cases} \tag{2}$$

Function ufp() is piecewise constant: the result of ufp(x) for $|x| \in [2^n, 2^{n+1})$ is the constant 2^n. Using this definition, the gap xe_x between the real and its floating-point representation can be rewritten as $\mathrm{ufp}(x)\, e_x$ and the rounding operator is now:

$$\mathrm{rnd}(x) = x + \mathrm{ufp}(x)\, e_x + d_x \tag{3}$$

Absolute and relative elementary rounding errors. We now define $\Gamma_a(x)$ and $\Gamma_r(x)$ the *elementary absolute* and *relative* rounding errors which occur when a real number x is rounded to its floating-point approximation $\widetilde{x} = \mathrm{rnd}(x)$:

$$\Gamma_a(x) = \mathrm{rnd}(x) - x = \mathrm{ufp}(x)\, e_x + d_x \tag{4}$$

The relative error is defined only when $x \neq 0$:

$$\Gamma_r(x) = \frac{\mathrm{rnd}(x) - x}{x} = \frac{\mathrm{ufp}(x)\, e_x + d_x}{x} \tag{5}$$

2.2 Arithmetic operations

The IEEE-754 norm standardises some operations that are required to be exactly rounded (addition, subtraction, multiplication, division and square root): the result of the floating-point operation on real operands is the same as if the operation was performed in real numbers on the given inputs, then rounded. For every operation $op : \mathbb{R}^k \to \mathbb{R}$ defined as *exactly* rounded, the corresponding floating-point operation $\widetilde{op}$ can be expressed as:

$$\widetilde{op}(x_1, \ldots, x_k) = \mathrm{rnd}(op(x_1, \ldots, x_k)) \tag{6}$$

The IEEE754-2008 revision additionally recommends that fifty additional operators are correctly rounded. We do not handle these operations in this work, but the approach developed here can be extended.

We now consider the propagation of errors through successive operations. We denote by $\widetilde{x}$ the approximation of an idealized computation x. We thus define the absolute error due to the approximation by:

$$\mathcal{E}_a(x) = \widetilde{x} - x$$

and the relative error, for $x \neq 0$, by:

$$\mathcal{E}_r(x) = \frac{\widetilde{x} - x}{x}$$

The absolute error on the result of an operation op on values $\widetilde{x_1}, \ldots, \widetilde{x_k}$ which are already the approximations of some idealized values $x_1, \ldots, x_k$ is defined by:

$$\mathcal{E}_a(op(x_1, \ldots, x_k)) = \widetilde{op}(\widetilde{x_1}, \ldots, \widetilde{x_k}) - op(x_1, \ldots, x_k)$$

where for all $i = 1, \ldots, k$, the approximated value $\widetilde{x_i}$ is such that $\widetilde{x_i} = x_i + \mathcal{E}_a(x_i) = x_i(1 + \mathcal{E}_r(x_i))$.

Using Equations (6) and (4), this can be rewritten:

$$\mathcal{E}_a(op(x_1, \ldots, x_k)) = op(\widetilde{x_1}, \ldots, \widetilde{x_k}) + \Gamma_a(op(\widetilde{x_1}, \ldots, \widetilde{x_k})) - op(x_1, \ldots, x_k) \quad (7)$$

The relative error is derived, when $op(x_1, \ldots, x_k) \neq 0$:

$$\mathcal{E}_r(op(x_1, \ldots, x_k)) = \frac{\mathcal{E}_a(op(x_1, \ldots, x_k))}{op(x_1, \ldots, x_k)} \quad (8)$$

2.3 Concrete semantics

The concrete model is that of traditional numerical error analyzes, and in particular the static analysis [9], which describe the difference of behavior between the execution of a program in real numbers and in floating-point numbers, *along the floating-point execution flow*. We consider in this work the analysis of a language with the operations $\{+, -, \times, \div, \sqrt{}\}$, which are required to be exactly rounded in the IEEE-754 standard, conditional statements and loops. The concrete value that we will compute for all program variables and control points of a program in this language, is $(x, \widetilde{x}, \mathcal{E}_a(x), \mathcal{E}_r(x))$, where:

- $\widetilde{x}$ is the result of the execution of the program in a floating-point semantics, until the control point of interest,
- x is the result of the execution of the same sequence of arithmetic operations in a real semantics, ignoring the possibility of a control flow divergence due to rounding errors,
- the errors between the real execution and the floating-point executions $\mathcal{E}_a(x)$ and $\mathcal{E}_r(x)$ are defined by Equations (7) and (8).

Conditional statements and unstable tests. In this work, the path conditions are those of the floating-point executions. We thus ignore the possibility of unstable tests, when for same input values, the floating-point and the real-valued executions can take different branches of a conditional statement. We simply issue a warning when this possibility is detected, as in for instance early versions of Fluctuat [7, 9]. In case an unstable test actually occurs, the analysis is possibly unsound, as the discontinuity error between the computations performed in the two branches should be considered as an additional error. Relational analyzes are needed to estimate such discontinuity errors in a not overly conservative way, and this has been studied and implemented for instance in [10, 5]. But the problem is somewhat orthogonal to the interplay between relative and absolute error considered here, and is also not considered in the most closely related work [16, 11], to which we compare our analysis in the section dedicated to experiments. But we intend to handle unstable tests in the future, in a relational version of the present analysis.

3 Interval-based Abstraction

Intervals [13] are used in many situations to rigorously compute with interval domains instead of reals. Throughout the paper, intervals are typeset in boldface letters. Let $\boldsymbol{x} = [\underline{x}, \overline{x}]$ be such an interval, with its bounds $\underline{x} \leq \overline{x}$ where $\underline{x} \in \mathbb{R} \cup \{-\infty\}$ and $\overline{x} \cup \{+\infty\}$. Interval arithmetic computes a bounding interval for each elementary operation by $\boldsymbol{x} \circ \boldsymbol{y} = [\min_{x \in \boldsymbol{x}, y \in \boldsymbol{y}} \{x \circ y\}, \max_{x \in \boldsymbol{x}, y \in \boldsymbol{y}} \{x \circ y\}]$, where $\circ \in \{+, -, \times, \div\}$, and analogously for the square root. Intervals are the basis of one of the first and most widely used numerical abstract domains, the lattice of intervals [3].

In what follows, we propose an abstraction which relies on the lattice of intervals: we abstract with intervals $(\widetilde{\boldsymbol{x}}, \boldsymbol{\mathcal{E}_a(x)}, \boldsymbol{\mathcal{E}_r(x)})$, the floating-point range, absolute and relative errors. The errors are computed, on each control-flow path, under the assumption for the error estimation that the real and floating-point executions follow the same path. We deduce bounds for the value in real-valued semantics by $\boldsymbol{x} = \widetilde{\boldsymbol{x}} - \boldsymbol{\mathcal{E}_a(x)}$. The abstract domain forms a complete lattice, fully relying on the lattice of intervals, with a join operator performed componentwise on the value and errors using the classical join operator on intervals.

The rounding mode for computing the interval extremities on the intervals bounding the floating-point range will be the rounding mode of the computation we analyse (rounding to the nearest). The other terms, that bound the errors and the real-valued range, will be computed with outward rounding, in order to ensure a sound implementation.

3.1 Abstraction of the elementary rounding errors.

In this section, we define the abstraction $\mathit{\Gamma}_a(\boldsymbol{x})$ and $\mathit{\Gamma}_r(\boldsymbol{x})$ of the elementary rounding errors defined by (4) and (5). They will be used for the abstraction of transfer functions in Section 3.2.

Terms e_x and d_x that appear in the elementary rounding errors are bounded respectively in $[-\epsilon_M, \epsilon_M]$ and $[-\delta_M, \delta_M]$. Additionally, we know that e_x and d_x cannot be both non-zero for the same x. If x is rounded to a normalized number, then $d_x = 0$ and if x is rounded to a denormalized number, then $e_x = 0$. We can thus compute the abstraction of the elementary absolute rounding error over an interval of real numbers as the union of the two cases:

$$\mathit{\Gamma}_a(\boldsymbol{x}) = \mathrm{ufp}(\boldsymbol{x})\, \epsilon(\boldsymbol{x}) \cup \delta(\boldsymbol{x}) \tag{9}$$

where ϵ (resp. δ) returns the interval $[-\epsilon_M, \epsilon_M]$ (resp. $[-\delta_M, \delta_M]$) if its parameter contains at least a normalized (resp. denormalized) number and $[0, 0]$ otherwise. Moreover, as ufp() is increasing in the absolute value of its argument, we can abstract the rounding error on normalized numbers by

$$\mathrm{ufp}(\boldsymbol{x})\, \epsilon(\boldsymbol{x}) \subseteq \mathrm{ufp}(\max(|\underline{x}|, |\overline{x}|))\, \epsilon(\boldsymbol{x}).$$

Let us define $\mathrm{norm}(\boldsymbol{x})$ and $\mathrm{denorm}(\boldsymbol{x})$ that return respectively the subsets of normalized and denormalized numbers from interval $\boldsymbol{x}$. We can define the

abstraction $\Gamma_r(\boldsymbol{x})$ of the elementary relative error, for any interval $\boldsymbol{x}$, as:

$$\Gamma_r(\boldsymbol{x}) = \max_{x \in \text{norm}(\boldsymbol{x}), x \neq 0} \left| \frac{\text{ufp}(x)}{x} \right| [-\epsilon_M, \epsilon_M] \cup \max_{x \in \text{denorm}(\boldsymbol{x})} \frac{[-\delta_M, \delta_M]}{|x|} \tag{10}$$

Equation (10) will be used to derive in Section 3.2 relative error bounds also when interval $\boldsymbol{x}$ possibly contains zero. These error bounds will be valid whenever the relative error is defined, that is for all non zero value in $\boldsymbol{x}$.

Let us first evaluate in (10) the error due to the rounding of normalized numbers. Consider x strictly positive (the negative case is symmetric), then we can write:

$$\left| \frac{\text{ufp}(x)}{x} \right| = \frac{2^{exp}}{sig \times 2^{exp}} = \frac{1}{sig} \tag{11}$$

Given $sig \in [1, 2)$, a simple abstraction of $\left| \frac{\text{ufp}(x)}{x} \right|$ is the interval $\left(\frac{1}{2}, 1\right]$, and its maximum is always bounded by 1. However, we can slightly refine this estimate when there exists n such that $|\boldsymbol{x}| \subseteq [2^n, 2^{n+1})$. This gives, when $\boldsymbol{x}$ does not contain 0:

$$\max_{x \in \boldsymbol{x}} \left| \frac{\text{ufp}(x)}{x} \right| = \begin{cases} 1/sig_{\min(|\underline{x}|, |\overline{x}|)} & \text{if } \exists n \in \mathbb{Z}, |\boldsymbol{x}| \subseteq [2^n, 2^{n+1}) \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

Let us now consider the error due to denormalized numbers if $\boldsymbol{x}$ contains any. Let us consider again x strictly positive. A positive denormalized number can be expressed as a multiple of δ_M, i.e $x = n\delta_M$ with $n \in \mathbb{Z}$, and an absolute error of magnitude at most δ_M can be committed, we thus abstract the relative error on denormalized numbers by:

$$\max_{x \in \text{denorm}(\boldsymbol{x})} \frac{[-\delta_M, \delta_M]}{|x|} \subseteq [-1, 1] \tag{13}$$

3.2 Transfer Functions for Arithmetic Operations

Let us now study the transfer functions for each operation in $\{+, -, \times, \div, \sqrt{}\}$.

First, the floating-point range $\widetilde{\boldsymbol{x}}$ is abstracted classcially in interval arithmetic. Then the absolute and relative error bounds are computed as described in this section, by an interval abstraction of (7) and (8). Finally, bounds for the value in real-valued semantics are deduced by $\boldsymbol{x} = \widetilde{\boldsymbol{x}} - \boldsymbol{\mathcal{E}_a(x)}$.

Let us first state that after each operation, which yields a result $\boldsymbol{z} = op(\boldsymbol{x}, \boldsymbol{y})$, we perform a reduced product of the absolute and relative errors:

Reduction

$$\begin{cases} \mathcal{E}_a(\boldsymbol{z}) = \mathcal{E}_a(\boldsymbol{z}) \cap \mathcal{E}_r(\boldsymbol{z})\boldsymbol{z} \\ \mathcal{E}_r(\boldsymbol{z}) = \mathcal{E}_r(\boldsymbol{z}) \cap \dfrac{\mathcal{E}_a(\boldsymbol{z})}{\boldsymbol{z}} \text{ whenever } 0 \notin \boldsymbol{z} \end{cases} \tag{14}$$

We will see that, in particular for the division and the square root, the two types of errors are more tightly coupled than by the only use of this reduction.

Indeed, some formulations which are equivalent on real numbers, yield different levels of conservativeness when computed abstracted. It is thus important to carefully state the precise expression of the propagation of errors through arithmetic operations. We detail below, for each arithmetic operation, propagation rules that provide a sound abstraction, while reducing the wrapping effect due to the use of intervals:

Lemma 1 (Addition and Subtraction).

$$\mathcal{E}_a(\boldsymbol{x} \pm \boldsymbol{y}) = (\mathcal{E}_a(\boldsymbol{x}) \pm \mathcal{E}_a(\boldsymbol{y})) + \Gamma_a(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}}) \tag{15}$$

The relative error is defined only when $0 \notin \boldsymbol{x} \pm \boldsymbol{y}$*. In this case, we have:*

$$\mathcal{E}_r(\boldsymbol{x} \pm \boldsymbol{y}) = \begin{cases} \left(\dfrac{\mathcal{E}_r(\boldsymbol{x}) - \mathcal{E}_r(\boldsymbol{y})}{1 \pm \boldsymbol{y}/\boldsymbol{x}} + \mathcal{E}_r(\boldsymbol{y})\right)(1 + \Gamma_r(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}})) + \Gamma_r(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}}) \text{ if } 0 \notin \boldsymbol{x} \\ \left(\dfrac{\mathcal{E}_r(\boldsymbol{y}) - \mathcal{E}_r(\boldsymbol{x})}{1 \pm \boldsymbol{x}/\boldsymbol{y}} + \mathcal{E}_r(\boldsymbol{x})\right)(1 + \Gamma_r(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}})) + \Gamma_r(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}}) \text{ if } 0 \notin \boldsymbol{y} \end{cases} \tag{16}$$

When $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *both do not include zero, then the relative error can be computed as the intersection of the 2 estimates in (16).*

Proof. The propagation of the absolute error corresponds to a classical absolute rounding error analysis, starting from Equation (7) instantiated for the addition and subtraction: for all $x \in \boldsymbol{x}$, $y \in \boldsymbol{y}$, $\widetilde{x} = x + \mathcal{E}_a(x) \in \widetilde{\boldsymbol{x}}$, $\widetilde{y} = x + \mathcal{E}_a(y) \in \widetilde{\boldsymbol{y}}$,

$$\begin{aligned} \mathcal{E}_a(x \pm y) &= (\tilde{x} \pm \tilde{y}) + \Gamma_a(\tilde{x} \pm \tilde{y}) - (x \pm y) \\ &= ((x + \mathcal{E}_a(x)) \pm (y + \mathcal{E}_a(y))) + \Gamma_a(\tilde{x} \pm \tilde{y}) - (x \pm y) \\ &= (\mathcal{E}_a(x) \pm \mathcal{E}_a(y)) + \Gamma_a(\tilde{x} \pm \tilde{y}) \end{aligned}$$

Abstracting this result in intervals, we get Equation (15), which defines $\mathcal{E}_a(\boldsymbol{x} \pm \boldsymbol{y})$ as an interval over-approximation of $\{\mathcal{E}_a(x \pm y), x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$.

Note that we would naturally also get a sound over-approximation of $\mathcal{E}_a(\boldsymbol{x} \pm \boldsymbol{y})$ by directly computing in intervals $(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}}) + \Gamma_a(\widetilde{\boldsymbol{x}} \pm \widetilde{\boldsymbol{y}}) - (\boldsymbol{x} \pm \boldsymbol{y})$. However, the result would be very conservative in general, because interval arithmetic does not handle correlations. We thus derive, for each arithmetic operation, error formulas in real numbers by reorganizing terms in an equivalent expression but reducing variable repetitions. We then abstract the final expression in intervals.

For any binary operation $\circ$, for all $x \in \boldsymbol{x}$, $y \in \boldsymbol{y}$ such that $x \circ y \neq 0$, for all $\widetilde{x} = x + \mathcal{E}_a(x)$, $\widetilde{y} = y + Ea(y)$, we can compute the relative error as

$$\begin{aligned} \mathcal{E}_r(x \circ y) &= \frac{(\tilde{x} \circ \tilde{y}) + \Gamma_a(\tilde{x} \circ \tilde{y}) - (x \circ y)}{x \circ y} \\ &= \frac{(\tilde{x} \circ \tilde{y}) + (\tilde{x} \circ \tilde{y})\Gamma_r(\tilde{x} \circ \tilde{y}) - (x \circ y)}{x \circ y} \end{aligned}$$

from which we have:

$$\mathcal{E}_r(x \circ y) = \frac{\tilde{x} \circ \tilde{y}}{x \circ y}(1 + \Gamma_r(\tilde{x} \circ \tilde{y})) - 1 \tag{17}$$

We can deduce:

$$\begin{aligned}\mathcal{E}_r(x \pm y) &= \frac{x(\mathcal{E}_r(x)+1) \pm y(1+\mathcal{E}_r(y))}{x \pm y}(1+\Gamma_r(\tilde{x} \pm \tilde{y})) - 1 \\ &= \frac{x\mathcal{E}_r(x) \pm y\mathcal{E}_r(y)}{x \pm y}(1+\Gamma_r(\tilde{x} \pm \tilde{y})) + \Gamma_r(\tilde{x} \pm \tilde{y})\end{aligned}$$

It is interesting to reformulate this expression in order to suppress as much as possible variable repetitions. This will reduce the loss of correlation when the expression will be evaluated in interval arithmetic. For $x \neq 0$, we can write:

$$\begin{aligned}\mathcal{E}_r(x \pm y) &= \left(\frac{x\mathcal{E}_r(x) - x\mathcal{E}_r(y)}{x \pm y} + \frac{x\mathcal{E}_r(y) \pm y\mathcal{E}_r(y)}{x \pm y}\right)(1+\Gamma_r(\tilde{x} \pm \tilde{y})) + \Gamma_r(\tilde{x} \pm \tilde{y}) \\ &= \left(\frac{\mathcal{E}_r(x) - \mathcal{E}_r(y)}{1 \pm \frac{y}{x}} + \mathcal{E}_r(y)\right)(1+\Gamma_r(\tilde{x} \pm \tilde{y})) + \Gamma_r(\tilde{x} \pm \tilde{y})\end{aligned}$$

A symmetric transformation can be done, exchanging x and y, which allows us to conclude with Equations (16), after abstraction in intervals.

We note that the addition is the operation for which propagating relative error is less natural, and thus some accuracy loss can be expected.

Lemma 2 (Multiplication).

$$\mathcal{E}_a(\boldsymbol{x} \times \boldsymbol{y}) = \boldsymbol{x}\mathcal{E}_a(\boldsymbol{y}) + \boldsymbol{y}\mathcal{E}_a(\boldsymbol{x}) + \mathcal{E}_a(\boldsymbol{x})\mathcal{E}_a(\boldsymbol{y}) + \Gamma_a(\widetilde{\boldsymbol{x}} \times \widetilde{\boldsymbol{y}}) \tag{18}$$

$$\mathcal{E}_r(\boldsymbol{x} \times \boldsymbol{y}) = (\mathcal{E}_r(\boldsymbol{x})+1)(\mathcal{E}_r(\boldsymbol{y})+1)(1+\Gamma_r(\widetilde{\boldsymbol{x}} \times \widetilde{\boldsymbol{y}})) - 1 \tag{19}$$

Proof. As for the addition, the expression of the propagated absolute error by the multiplication is quite natural, and corresponds to a classical absolute rounding error analysis: for all $x \in \boldsymbol{x}$, $y \in \boldsymbol{y}$, $\tilde{x} = x + \mathcal{E}_a(x)$, $\tilde{y} = y + \mathcal{E}_a(y)$,

$$\begin{aligned}\mathcal{E}_a(x \times y) &= (\tilde{x} \times \tilde{y}) + \Gamma_a(\tilde{x} \times \tilde{y}) - (x \times y) \\ &= (x + \mathcal{E}_a(x))(y + \mathcal{E}_a(y)) + \Gamma_a(\tilde{x} \times \tilde{y}) - (x \times y) \\ &= x\mathcal{E}_a(y) + y\mathcal{E}_a(x) + \mathcal{E}_a(x)\mathcal{E}_a(y) + \Gamma_a(\tilde{x} \times \tilde{y})\end{aligned}$$

Starting from Equation (17), we obtain an expression of the propagated relative error that naturally involves the relative errors on the operands:

$$\begin{aligned}\mathcal{E}_r(x \times y) &= \frac{\tilde{x} \times \tilde{y}}{x \times y}(1+\Gamma_r(\tilde{x} \times \tilde{y})) - 1 \\ &= \frac{x(\mathcal{E}_r(x)+1) \times y(\mathcal{E}_r(y)+1)}{x \times y}(1+\Gamma_r(\tilde{x} \times \tilde{y})) - 1 \\ &= (\mathcal{E}_r(x)+1)(\mathcal{E}_r(y)+1)(1+\Gamma_r(\tilde{x} \times \tilde{y})) - 1\end{aligned}$$

This propagation of relative errors should be quite accurate, as we could remove correlations to values of x and y.

Example 3. We consider a very simple example to examplify our analysis. We have 4 input floating-point variables `a, b, c, d` given respectively in the ranges [0,1], [1,2], [0,1], and [1,2]. All these inputs are supposed to be known exactly, with no rounding error, i.e. $\forall \boldsymbol{v} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}, \mathcal{E}_a(\boldsymbol{v}) = \mathcal{E}_r(\boldsymbol{v}) = [0, 0]$. We now consider the errors committed on the computations that define floating-point variables `x, y, z`. For the sake of demonstration, we explicit here the computation of errors in term of the machine epsilon ϵ_M. In order to evaluate the error on variable `x`, we first evaluate `3b`: its range of value is $3\boldsymbol{b} = [3, 6]$, and the errors are $\mathcal{E}_r(3\boldsymbol{b}) = \Gamma_r(3\boldsymbol{b}) = [-\epsilon_M, +\epsilon_M]$ and $\mathcal{E}_a(3\boldsymbol{b}) = \Gamma_a(3\boldsymbol{b}) = [-4\epsilon_M, +4\epsilon_M]$ The errors on `x` that results from the addition are obtained using (15) and (16):

```
x = a + 3 * b ;
y = c + 3 * d ;
z = x * y ;
```

$$\mathcal{E}_a(\boldsymbol{x}) = \mathcal{E}_a(3\boldsymbol{b}) + \Gamma_a([3, 7]) = [-8\epsilon_M, +8\epsilon_M]$$

$$\begin{aligned}\mathcal{E}_r(\boldsymbol{x}) &= \left(\frac{\mathcal{E}_r(3\boldsymbol{b}) - \mathcal{E}_r(\boldsymbol{a})}{1 + a/3\boldsymbol{b}} + \mathcal{E}_r(\boldsymbol{a})\right)(\Gamma_r([3, 7]) + 1) + \Gamma_r([3, 7]) \\ &= \frac{[-\epsilon_M, +\epsilon_M]}{[1, 4/3]}([-\epsilon_M, +\epsilon_M] + 1) + [-\epsilon_M, +\epsilon_M] = [-2\epsilon_M, +2\epsilon_M] + O(\epsilon_M^2)\end{aligned}$$

Note that the bounds for the relative errors are $\frac{4}{3}$ times better than the a posteriori estimate $\mathcal{E}_a(\boldsymbol{x})/\boldsymbol{x} = \left[-\frac{8}{3}\epsilon_M, +\frac{8}{3}\epsilon_M\right]$. The errors on `y` are computed similarly. Finally, we can deduce absolute error bounds for `z` using (18):

$$\begin{aligned}\mathcal{E}_a(\boldsymbol{z}) &= \boldsymbol{x}\mathcal{E}_a(\boldsymbol{y}) + \boldsymbol{y}\mathcal{E}_a(\boldsymbol{x}) + \mathcal{E}_a(\boldsymbol{x})\mathcal{E}_a(\boldsymbol{y}) + \Gamma_a(\boldsymbol{x} \times \boldsymbol{y}) \\ &= 2 \times [3, 7]\,[-8\epsilon_M, +8\epsilon_M] + [-8\epsilon_M, +8\epsilon_M]\,[-8\epsilon_M, +8\epsilon_M] + \Gamma_a([9, 49]) \\ &= [-144\epsilon_M, +144\epsilon_M] + O(\epsilon_M^2)\end{aligned}$$

and relative error bounds using (19):

$$\begin{aligned}\mathcal{E}_r(\boldsymbol{z}) &= (\mathcal{E}_r(\boldsymbol{x}) + 1)(\mathcal{E}_r(\boldsymbol{y}) + 1)(\Gamma_r(\boldsymbol{x} \times \boldsymbol{y}) + 1) - 1 \\ &= ([-2\epsilon_M, +2\epsilon_M] + 1)([-2\epsilon_M, +2\epsilon_M] + 1)(\Gamma_r([9, 49]) + 1) - 1 + O(\epsilon_M^2) \\ &= [-5\epsilon_M, +5\epsilon_M] + O(\epsilon_M^2)\end{aligned}$$

Neglecting here for simplicity the second order errors, the reduced product yields an estimate for the relative error approximately 3.2 times better than the a posteriori estimate $\mathcal{E}_a(\boldsymbol{z})/[9, 49]$.

Lemma 3 (Division). *The division is defined whenever $0 \notin y$, by:*

$$\mathcal{E}_a(\boldsymbol{x} \div \boldsymbol{y}) = \frac{\mathcal{E}_a(\boldsymbol{x}) - \boldsymbol{x}\mathcal{E}_r(\boldsymbol{y})}{\widetilde{\boldsymbol{y}}} + \Gamma_a(\widetilde{\boldsymbol{x}} \div \widetilde{\boldsymbol{y}}) \tag{20}$$

$$\mathcal{E}_r(\boldsymbol{x} \div \boldsymbol{y}) = \frac{\mathcal{E}_r(\boldsymbol{x}) + 1}{\mathcal{E}_r(\boldsymbol{y}) + 1}(1 + \Gamma_r(\widetilde{\boldsymbol{x}} \div \widetilde{\boldsymbol{y}})) - 1 \tag{21}$$

Proof. The division is a case where the coupling between the computation of two errors is integrated: the absolute error on $x \div y$ naturally involves the absolute error on x and the relative error on y:

$$\begin{aligned}
\mathcal{E}_a(x \div y) &= (\tilde{x} \div \tilde{y}) + \Gamma_a(\tilde{x} \div \tilde{y}) - (x \div y) \\
&= \frac{y\tilde{x} - \tilde{y}x}{y\tilde{y}} + \Gamma_a(\tilde{x} \div \tilde{y}) \\
&= \frac{y(x + \mathcal{E}_a(x)) - (y + \mathcal{E}_a(y))x}{y\tilde{y}} + \Gamma_a(\tilde{x} \div \tilde{y}) \\
&= \frac{\mathcal{E}_a(x) - x\mathcal{E}_r(y)}{\tilde{y}} + \Gamma_a(\tilde{x} \div \tilde{y})
\end{aligned}$$

This is thus an operation where propagating tight relative error bounds on the operands proves useful to tighten the absolute error bounds on the result.

$$\begin{aligned}
\mathcal{E}_r(x \div y) &= \frac{\tilde{x} \div \tilde{y}}{x \div y}(1 + \Gamma_r(\tilde{x} \div \tilde{y})) - 1 \\
&= \frac{x(\mathcal{E}_r(x) + 1)}{y(\mathcal{E}_r(y) + 1)} \times \frac{y}{x} \times (1 + \Gamma_r(\tilde{x} \div \tilde{y})) - 1 \\
&= \frac{\mathcal{E}_r(x) + 1}{\mathcal{E}_r(y) + 1}(1 + \Gamma_r(\tilde{x} \div \tilde{y})) - 1
\end{aligned}$$

As for the multiplication, we note that the propagation of relative errors should be generally quite accurate. Note also that as for the multiplication as well, the relative error bounds are defined even when $0 \in \boldsymbol{x} \div \boldsymbol{y}$, as long as they are defined for $\boldsymbol{x}$ and $\boldsymbol{y}$. The bound will be valid for all nonzero values in $\boldsymbol{x} \div \boldsymbol{y}$. This is exemplified in Example 4.

Example 4. Let us come back to Example 2 of the introduction. An error is committed when computing $t+1$, the absolute error of $t+1$ is bounded by $\mathcal{E}_a(t+1) = [-5.68e^{-14}, 5.68e^{-14}]$, the relative error by $\mathcal{E}_r(t+1) = [-1.1e^{-16}, 1.1e^{-16}]$. Using Equation (20) to bound the absolute error of the division, we obtain:

$$\begin{aligned}
\mathcal{E}_a\left(\frac{\boldsymbol{t}}{\boldsymbol{t}+1}\right) &= -\frac{\boldsymbol{t}\mathcal{E}_r(\boldsymbol{t}+1)}{\widetilde{\boldsymbol{t}}+1} + \Gamma_a\left(\frac{\widetilde{\boldsymbol{t}}}{\widetilde{\boldsymbol{t}}+1}\right) \\
&= \frac{[0, 900][-1.1e^{-16}, 1.1e^{-16}]}{[1, 1000]} + [-5.68e^{-14}, 5.68e^{-14}] \\
&= [-1.67e^{-13}, 1.67e^{-13}]
\end{aligned}$$

If only absolute error bounds were available, we would replace $\boldsymbol{x}\mathcal{E}_r(\boldsymbol{y})$ by $\boldsymbol{x}\mathcal{E}_a(\boldsymbol{y})/\boldsymbol{y}$ in Equation (20) and obtain the absolute error analysis used classically. We would then obtain as absolute error bound on $\boldsymbol{t}/(\boldsymbol{t}+\boldsymbol{1})$:

$$\frac{[0, 999] \times [-5.68e^{-14}, 5.68e^{-14}]}{[1, 1000]^2} + [-5.68e^{-14}, 5.68e^{-14}] = [-5.68e^{-11}, 5.68e^{-11}]$$

which is 340 times larger than the absolute error bound computed using our reduced product.

The relative error on $\boldsymbol{t}/(\boldsymbol{t}+\mathbf{1})$ is bounded where it is defined, that if for all $t \neq 0$, by:

$$\begin{aligned}\mathcal{E}_r\left(\frac{\boldsymbol{t}}{\boldsymbol{t}+1}\right) &= \frac{\mathcal{E}_r(\boldsymbol{t})+1}{\mathcal{E}_r(\boldsymbol{t}+\mathbf{1})+1}\left(1+\Gamma_r\left(\frac{\tilde{t}}{\tilde{t}+1}\right)\right)-1 \\ &= \frac{1}{[-1.1e^{-16},1.1e^{-16}]+1}\left(1+\Gamma_r\left(\frac{\tilde{t}}{\tilde{t}+1}\right)\right)-1\end{aligned}$$

For $\boldsymbol{t}=[0,999]$, $\boldsymbol{t}$ and $\boldsymbol{t}/(\boldsymbol{t}+\mathbf{1})$ contain zero. As t is an input, its relative error $\mathcal{E}_r(\boldsymbol{t})$ is zero when it is defined, that is for all $t \neq 0$. The elementary relative error Γ_r is also defined for all $t \neq 0$, and is bounded by the maximum relative error when using denormalized floating-point numbers around 0, given by (13): for all $t \neq 0,\ \Gamma_r(\frac{t}{t+1}) \subseteq [-1,1]$. Thus

$$\mathcal{E}_r\left(\frac{\boldsymbol{t}}{\boldsymbol{t}+1}\right) = \frac{1}{[-1.1e^{-16},1.1e^{-16}]+1}\left(1+[-1,1]\right)-1 \quad \subseteq 1+[-1,1]-1=[-1,1]$$

The square root is an operation where the relative error on the operand naturally appears in the propagation:

Lemma 4 (Square root).

$$\mathcal{E}_a(\sqrt{\boldsymbol{x}}) = \sqrt{\boldsymbol{x}}(\sqrt{1+\mathcal{E}_r(\boldsymbol{x})}-1)+\Gamma_a(\sqrt{\tilde{\boldsymbol{x}}}) \tag{22}$$

$$\mathcal{E}_r(\sqrt{\boldsymbol{x}}) = \sqrt{1+\mathcal{E}_r(\boldsymbol{x})}(\Gamma_r(\sqrt{\tilde{\boldsymbol{x}}})+1)-1 \tag{23}$$

Proof. In order to avoid the loss of correlation, it is natural to factorize $\sqrt{x}$:

$$\begin{aligned}\mathcal{E}_a(\sqrt{x}) &= \sqrt{\tilde{x}}+\Gamma_a(\sqrt{\tilde{x}})-\sqrt{x} \\ &= \sqrt{x}(\sqrt{1+\mathcal{E}_r(x)}-1)+\Gamma_a(\sqrt{\tilde{x}})\end{aligned}$$

The expression of the relative error is deduced immediately.

$$\begin{aligned}\mathcal{E}_r(\sqrt{x}) &= \frac{\sqrt{\tilde{x}}}{\sqrt{x}}(\Gamma_r(\sqrt{\tilde{x}})+1)-1 \\ &= \sqrt{\frac{x(\mathcal{E}_r(x)+1)}{x}}(\Gamma_r(\sqrt{\tilde{x}})+1)-1 \\ &= \sqrt{\mathcal{E}_r(x)+1}(\Gamma_r(\sqrt{\tilde{x}})+1)-1\end{aligned}$$

Using similar developments, it is possible to handle more functions. We chose in this work to focus on the main arithmetic operations which error bounds have long been specified by the IEEE 754 norm.

3.3 Handling conditional statements

Interpretation of Conditional expressions. Let $\gamma(x_1, \ldots, x_n)$ a conditional expression defined by $f(x_1, \ldots, x_n) \diamond b$, with $\diamond \in \{<, >, =, \leq, \geq\}$. Let us denote by $\widetilde{\boldsymbol{x}}_\gamma$ the interval abstraction of the floating-point value of variable x, after transformation by the interpretation of conditional γ. It is computed by the classical backward constraint propagation on intervals which filters out values of $\widetilde{\boldsymbol{x}}$ that do not satisfy the constraint. Note that as already discussed in the section devoted to the concrete semantics, we consider the path condition only on the floating-point value. The bound on the relative error is left unchanged by the interpretation of constraints:

$$\mathcal{E}_r(\boldsymbol{x}_\gamma) = \mathcal{E}_r(\boldsymbol{x}) \tag{24}$$

In a classical error analysis, that is, with no information about the relative error, the absolute error bounds are also left unchanged by the interpretation of this conditional: $\mathcal{E}_a(\boldsymbol{x}_\gamma) = \mathcal{E}_a(\boldsymbol{x})$. When available, the relative error bounds can be used to reduce the absolute error in the case the range of values $\boldsymbol{x}_\gamma$ has been reduced compared to $\boldsymbol{x}$ by the constraint propagation:

$$\mathcal{E}_a(\boldsymbol{x}_\gamma) = \mathcal{E}_a(\boldsymbol{x}) \cap \boldsymbol{x}_\gamma \mathcal{E}_r(\boldsymbol{x}). \tag{25}$$

Example 5. In Example 1 of the introduction, the multiplication `x:=i*i` results in $\boldsymbol{x} = [1, 10000]$ with an elementary absolute rounding error $\mathcal{E}_a(x) = [-9.09e^{-13}, 9.09e^{-13}]$, and a relative error bound $\mathcal{E}_r(x) = [-1.11e^{-16}, 1.11e^{-16}]$. The constraint `x <=2` yields $\boldsymbol{x}_{(\boldsymbol{x} \leq \mathbf{2})} = [1, 2]$ in the `true` branch. We can then reduce the absolute error bound on `x` in this branch (and thus on `y`), by $\mathcal{E}_a(\boldsymbol{y}) = [1, 2][-1.11e{-}16, 1.11e^{-16}]$, as already stated in this introductory example.

Join and widening Joining values coming from different branches of the program analyzed supposes to define a join or upper bound operator on abstract values. The join can be performed componentwise on the value, relative and absolute errors, relying on the classical join on intervals. Naturally, this relies on the hypothesis that there is no unstable test. Similarly, a widening can be defined componentwise relying on any widening operator on intervals.

4 Implementation and Experimental Evaluation

We have implemented this approach as a new abstract domain called Numerors in the Abstract Interpretion plug-in Eva of the verification platform Frama-C[4]. Frama-C provides a collection of plug-ins that perform static analysis, deductive verification, and testing, for safety- and security-critical software. Those plug-ins can cooperate thanks to their integration on top of a shared kernel and data structures along with their compliance to a common specification language.

[4] Our abstract domain should be included in an upcoming release of Frama-C/Eva (https://frama-c.com/value.html)

In what follows, we evaluate our approach by comparing the error bounds obtained by our tool against the state of the art tools Fluctuat [9], Daisy [11] and FPTaylor [16], on a set of representative benchmark examples.

Name	**Under-Apprx**	**Numerors**	**Fluctuat** Intervals	**Fluctuat** Affine	**Daisy** 1	**Daisy** 2	**FPTaylor**
log_approx	–	**6.25e-14**	*3.56e-11*	*3.56e-11*	–	–	–
conditional_ex	–	**2.22e-16**	*9.09e-13*	*9.09e-13*	–	–	*9.09e-13*
conditional_1	–	**8.43e-13**	*6.82e-12*	*6.82e-12*	–	–	2.09e-11
sqrt_1	2.11e-16	5.51e-15	3.72e-14	3.38e-14	3.72e-14	*4.52e-16*	**2.75e-16**
complex_sqrt	5.00e-16	*1.29e-15*	3.93e-15	2.52e-15	3.92e-15	1.89e-15	**5.70e-16**
kepler0	2.42e-13	*3.63e-13*	*3.63e-13*	*3.63e-13*	*3.63e-13*	7.15e-13	**3.18e-13**
intro_example	1.65e-16	1.68e-13	5.68e-11	5.67e-11	5.68e-11	*2.52e-16*	**1.67e-16**
sec4_example	3.25e-15	6.35e-11	1.16e-09	1.16e-09	1.16e-09	**7.00e-14**	*3.73e-13*
test01_sum3	8.88e-16	*3.33e-15*	*3.33e-15*	**2.89e-15**	*3.33e-15*	4.11e-15	**2.89e-15**
test02_sum8	4.00e-15	**6.22e-15**	**6.22e-15**	**6.22e-15**	**6.22e-15**	*9.55e-15*	**6.22e-15**
test03_nonlin2	1.64e-16	3.11e-15	2.42e-14	2.28e-14	2.42e-14	*4.45e-16*	**3.47e-16**
test04_dqmom9	5.87e-12	8.64e-05	8.64e-05	8.64e-05	8.64e-05	**1.78e-09**	*1.85e-05*
test05_nonlin1_r4	1.32e-12	2.78e-07	1.67e-06	1.67e-06	1.67e-06	**5.93e-11**	*2.21e-09*
test05_nonlin1_test2	8.29e-17	**8.33e-17**	**8.33e-17**	**8.33e-17**	**8.33e-17**	*1.39e-16*	**8.33e-17**
doppler1	6.13e-14	*1.62e-13*	3.45e-13	3.45e-13	3.91e-13	1.74e-13	**9.91e-14**
doppler2	1.14e-13	3.27e-13	8.78e-13	8.78e-13	9.78e-13	*3.18e-13*	**1.84e-13**
doppler3	4.16e-14	*8.50e-14*	1.36e-13	1.36e-13	1.60e-13	9.13e-14	**5.70e-14**
rigidBody1	1.79e-13	*2.40e-13*	*2.40e-13*	*2.40e-13*	*2.40e-13*	5.08e-13	**2.13e-13**
rigidBody2	1.81e-11	*2.31e-11*	*2.31e-11*	*2.31e-11*	*2.31e-11*	6.32e-11	**2.27e-11**
turbine1	4.30e-15	4.73e-14	6.04e-14	5.76e-14	6.04e-14	*2.80e-14*	**1.24e-14**
turbine2	4.41e-15	8.57e-15	8.57e-15	*8.54e-15*	8.57e-15	1.71e-14	**7.38e-15**
turbine3	3.22e-15	3.85e-14	4.72e-14	4.54e-14	4.72e-14	*1.65e-14*	**7.15e-15**
verhulst	1.70e-16	3.77e-16	3.77e-16	*3.00e-16*	3.80e-16	4.21e-16	**1.79e-16**
predatorPrey	8.79e-17	1.40e-16	1.40e-16	*1.38e-16*	1.41e-16	2.27e-16	**1.01e-16**
carbonGas	3.13e-09	2.00e-08	2.00e-08	1.58e-08	2.06e-08	*1.03e-08*	**4.96e-09**
sine	2.71e-16	*5.18e-16*	*5.18e-16*	*5.18e-16*	*5.18e-16*	6.55e-16	**4.38e-16**
sqroot	4.41e-16	*5.62e-16*	*5.62e-16*	*5.62e-16*	*5.62e-16*	7.89e-16	**4.86e-16**
sineOrder3	3.11e-16	5.91e-16	5.96e-16	*5.86e-16*	7.84e-16	7.99e-16	**5.28e-16**

Table 1. Tools Comparison on the Absolute Errors

Selection and description of the benchmarks. The examples are mostly extracted from the FPBench[5] suite for comparing verification tools on floating-point programs, to which we have added 4 examples of our own. Our selection has been guided by the will to keep a reasonably small set of examples, while including most classes of examples which were previously studied with the tools of the related work, Daisy and FPTaylor. We excluded the examples containing calls to mathematical functions like transcendentals that we do not handle, variations of the same examples that did not show a different behavior, and examples for which the inputs were not fully specified. Finally, we modified[6] the inputs of some examples so that all tools compute a non trivial relative error bounds.

The first four examples were written to highlight some features that were not well represented in FPBench, and in particular programs that include some

[5] http://fpbench.org

[6] http://www.lix.polytechnique.fr/Labo/Maxime.Jacquemin/numerors.c for the examples.

conditional statements (the first three examples), and a program with square roots (the fourth). Benchmark *log_approx* computes an approximation of the logarithm of the square of its input, using a loop and a Taylor expansion. Benchmark *conditional_ex* is Example 1, and *conditional_1* is similar but with more computation. Finally, *sqrt_1* computes the function $\sqrt{2x+3}/(2\sqrt{x}+3)$. The remaining examples come from the FPbench suite. Example *complex_sqrt* belongs to the Herbie [14] suite. Examples from *intro_example* to *test05_nonlin1_test2* come from the FPTaylor test suite, and *intro_example* and *sec4_example* are specifically used in [16] to introduce their technique. The remaining examples come either from the Rosa [5] or the FPTaylor test suites, and have already been used as benchmarks for both absolute and relative errors [16, 11].

Name	**Under-Approx**	**Numerors**	**Posteriori**	**Daisy** 1	**Daisy** 2	**Daisy** 3	**FPTaylor**
log_approx	–	∞	∞	–	–	–	–
conditional_ex	–	**1.11e-16**	9.09e-13	–	–	–	*1.13e-16*
conditional_1	–	*8.94e-16*	3.41e-12	–	–	–	**7.22e-16**
sqrt_1	3.42e-16	*6.61e-16*	6.83e-12	1.10e-12	1.02e-15	∞	**4.26e-16**
complex_sqrt	2.04e-16	*4.98e-16*	8.64e-14	4.01e-15	1.94e-15	∞	**2.64e-16**
kepler0	3.65e-16	1.20e-15	1.20e-15	1.20e-15	2.13e-15	*1.06e-15*	**5.71e-16**
intro_example	1.87e-16	**1.00**	∞	∞	∞	∞	∞
sec4_example	6.52e-15	**1.40e-13**	8.71e-06	8.71e-06	3.60e-13	*2.34e-13*	6.65e-12
test01_sum3	2.78e-16	∞	∞	∞	*1.37e-15*	∞	**5.05e-16**
test02_sum8	3.42e-16	*5.94e-16*	7.77e-16	7.77e-16	1.19e-15	7.73e-16	**4.82e-16**
test03_nonlin2	2.17e-16	∞	∞	∞	∞	∞	∞
test04_dqmom9	1.46e-12	∞	∞	∞	∞	∞	∞
test05_nonlin1_r4	2.63e-12	**5.55e-12**	5.00e-01	5.00e-01	*2.07e-10*	1.68e-05	3.46e-06
test05_nonlin1_test2	1.66e-16	*2.26e-16*	2.50e-16	2.50e-16	4.17e-16	2.96e-16	**1.69e-16**
doppler1	6.70e-16	*1.10e-15*	1.17e-11	1.33e-11	5.91e-12	1.26e-15	**9.69e-16**
doppler2	7.17e-16	*1.21e-15*	4.62e-11	5.14e-11	1.67e-11	1.37e-15	**9.13e-16**
doppler3	5.63e-16	*9.75e-16*	3.11e-13	3.65e-13	1.82e-13	1.14e-15	**7.36e-16**
rigidBody1	3.44e-16	*7.79e-16*	1.04e-12	1.04e-12	2.21e-12	9.76e-16	**4.39e-16**
rigidBody2	4.84e-16	*9.65e-16*	1.32e-15	1.32e-15	3.50e-15	1.17e-15	**6.27e-16**
turbine1	4.21e-16	3.05e-14	3.90e-14	3.90e-14	1.41e-14	*1.75e-15*	**7.95e-16**
turbine2	2.30e-16	*4.98e-16*	*4.98e-16*	*4.98e-16*	9.23e-16	6.92e-16	**3.97e-16**
turbine3	3.53e-16	7.50e-14	1.01e-13	1.01e-13	2.91e-14	*6.51e-15*	**2.40e-15**
verhulst	2.26e-16	*3.75e-16*	1.20e-15	1.21e-15	1.16e-15	4.59e-16	**2.41e-16**
predatorPrey	3.12e-16	*4.82e-16*	3.76e-15	3.77e-15	6.09e-15	6.87e-16	**3.58e-16**
carbonGas	3.39e-16	**7.16e-16**	9.52e-15	9.80e-15	2.40e-15	8.11e-16	*7.67e-16*
sine	2.71e-16	1.75e-15	2.27e-15	2.27e-15	8.56e-16	*6.31e-16*	**4.41e-16**
sqroot	3.99e-16	6.72e-16	6.72e-16	6.72e-16	7.91e-16	*5.66e-16*	**4.44e-16**
sineOrder3	3.53e-16	1.13e-14	1.41e-14	1.86e-14	9.11e-16	*8.94e-16*	**6.06e-16**

Table 2. Tools Comparison on the Relative Errors

Methodology of comparison. For each example, absolute error bounds on the output of interest are presented in Table 1, relative error bounds in Table 2. Table 3 presents the running times for each tool, on the complete set of benchmarks.

In Tables 1 and 2, underestimates of the errors are given in the first column denoted Under-Approx. They are obtained with Daisy, computing the maximum of errors obtained for runs on 100000 random input values. The second column denoted Numerors in all result tables presents the results of the approach presented in our work. Fluctuat in its Intervals mode is used in third column of

Table 1 as a witness of the results obtained with a classical interval absolute error analysis. The results of the corresponding a posteriori computation of the relative error are presented in third column of Table 2. This also corresponds to the results of our tool Numerors on absolute error, without the reducted product. In the fourth column of Table 1, we give the results of Fluctuat in its affine arithmetic based relational mode. We do not report results in Table 2, as the error is only computed a posteriori, and does not bring much new information. Finally, Daisy and FPTaylor are state on the art references for the computation of relative error bounds, relying on optimisation techniques. Daisy provides many possible options, with different evaluation modes both for values and errors, that combine interval and affine arithmetic based estimates with SMT. We included a representative set, providing different trade-offs between efficiency and accuracy. Mode Daisy 1 is:

```
daisy --analysis=dataflow --rangeMethod=intervals --errorMethod=interval
```

Mode Daisy 2 is

```
daisy --analysis=opt --rangeMethod=smt --errorMethod=affine.
```

Finally, mode Daisy 3, which is dedicated to relative error (and provides to absolute error bound, hence is not included in Table 1), is:

```
daisy --analysis=relative --rel-rangeMethod=smtcomplete.
```

Our understanding is that Daisy 3 corresponds to [11]. Note that Daisy does not currently handle conditional statements, so that we give no estimate for these examples. We can specify conditionals as constraints to FPTaylor, however this is unconvenient for loops, so that we do not give any estimate for *log_approx*.

Results. Let us first comment the timings results presented in Table 3, that correspond, for each tool, to the sum of times spent for each of the examples that could be analyzed. Fluctuat, which is dedicated to error analysis, is the fastest, even in its affine mode: the examples here being all quite simple, affine forms scale well. While our abstract domain should theoretically be comparable to Fluctuat Intervals, it is here 10 times slower, partially due to the less specialization of the frama-C core on which we rely. Finally and most importantly, we want to stress that Numerors is drastically more efficient than Daisy in any of its modes or than FPTaylor, which rely on SMT or optimization. We compare ourselves to the results given by these tools, not because we aim at being more accurate, but in order to demonstrate that we manage to often come close, or even beat in precision these more costly approaches, while keeping very low analysis costs.

Numerors	**Fluctuat** Intervals	**Fluctuat** Affine	**Daisy** 1	**Daisy** 2	**Daisy** 3	**FPTaylor**
0.271	0.059	0.049	4.220	652.062	16056.987	197.774

Table 3. Times Comparison

On each line, the result in boldface letters corresponds to the best estimate, the one in italic one corresponds to the second best one.

Numerors does not manage to bound the relative error on four examples: (*log_approx*, *test01_sum3*, *test03_nonlin2* and *test04_dqmom9*. The reason is that it finds that the range of the variable of interest includes zero for all these examples, and some addition and subtraction where involved with a result including zero. Daisy and FPTaylor also cannot bound the relative error on three of these examples, for the same reason. On the fourth (*test01_sum3*), Daisy and FPTaylor get finite bounds. This is because the actual range does not include zero, but a relational analysis is needed to infer this. Note that the reduced product would allow us to compute again relative error bounds on further computations (if they do not include zero in their result range). Finally, on example *intro_example*, which corresponds to Example 2, the range of the result also contains zero, and Numerors produces an error bound of 1 while Daisy and FPTaylor do not produce relative error bounds. This bound of 1 is valid for all non-zero values of the range, and corresponds to the maximum error bounds on denormalized numbers, as detailed in Example 4.

One obvious strength of our analysis concerns the interpretation of conditional statements. This is highlighted by the first three examples, where we are much better on absolute errors than FPTaylor, and comparable in relative error.

On the remaining benchmarks, our experiments often demonstrate that FPTaylor obtains the most accurate results, both in absolute and relative errors. However, for the absolute error, Numerors is still the best for 2 cases out of 25, and it is second best in more 10 cases. For the relative error, considering only the 23 examples for which at least one tool finds a finite bound, it is best on 4 examples, and sometimes even spectacularly so, and second best in 12 examples. Naturally, it is also always at least - and often considerably - better than Fluctuat in its interval mode. But it is also often better also than Fluctuat in its affine mode, when the loss of correlation due to intervals is not too important.

5 Conclusion and Future Work

We have demonstrated how a simple interval-based reduced product of absolute and relative error bounds greatly enhanced the absolute error bounds, at a very low additional cost. A possible additional interest of such analysis, is to further valorize these relative error bounds: indeed, some undesired phenomena of floating-point arithmetic, such as catastrophic cancellation, are best detected using relative error bounds.

Another possibility offered by this analyis is to refine error bounds by local subdivisions of the range of variables. Partitioning the range of some well chosen program variables is an approach used in most tools, that often allows them to considerably narrow value and error estimation. But this approach is costly, and it usually has to be performed globally on the program. We believe that relative error bounds can be used to enhance the quality of results using much cheaper local subdivisions, using an idea similar to the interpretation of conditional: the

relative error bounds can be used to reduce the absolute error bounds, given some aditional constraint on the value that corresponds to a local subdivision of the range. This refinement can be used to improve locally an error estimation, on demand. An additional advantage of the local compared to the global subdivision is that it can be realized on any quantity and not only on input variables.

Finally, a natural extension of this work, that we intend to investigate in the future, is its combination with relational abstractions such as affine forms. A simple way to do this is to simply adapt the analysis presented here, using a relational abstraction to bound the values and errors. But we expect that more refined interactions would improve the strength of the analysis. Using a relational abstraction will also allow us to accurately compute discontinuity errors between branches, and thus handle possible control flow divergences.

Acknowledgments

The work was partially supported by ANR project ANR-15-CE25-0002. We also gratefully acknowledge the help of the anonymous reviewers and Jérôme Féret, in improving the presentation of this work.

References

1. J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. *SEBGRAPI'93*, 1993.
2. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
3. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. ISOP'76*, pages 106–130. Dunod, Paris, France, 1976.
4. Eva Darulova and Viktor Kuncak. Trustworthy numerical computation in Scala. In *OOPSLA*, 2011.
5. Eva Darulova and Viktor Kuncak. Sound compilation of reals. POPL '14, pages 235–248, New York, NY, USA, 2014. ACM.
6. Marc Daumas and Guillaume Melquiond. Certification of bounds on expressions involving rounded operators. *ACM Trans. Math. Softw.*, 2010.
7. D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine. Towards an industrial use of Fluctuat on safety-critical avionics software. In *14th FMICS*, volume 5825 of *LNCS*, 2009.
8. E. Goubault. Static analyses of the precision of floating-point operations. In *8th Static Analysis International Symposium SAS'01*, volume 2126 of *Lecture Notes in Computer Science*, pages 234–259, 2001.
9. Eric Goubault and Sylvie Putot. Static analysis of finite precision computations. In *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2011.
10. Eric Goubault and Sylvie Putot. Robustness analysis of finite precision implementations. In *APLAS*, pages 50–57, 2013.
11. Anastasiia Izycheva and Eva Darulova. On sound relative error bounds for floating-point arithmetic. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, FMCAD '17, pages 15–22, 2017.

12. Victor Magron, George Constantinides, and Alastair Donaldson. Certified roundoff error bounds using semidefinite programming. Submitted.
13. Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. 2009.
14. J.-R. Wilcox P. Panchekha, A. Sanchez-Stern and Z. Tatlock. Automatically improving accuracy for floating point expressions. In D. Grove and S. Blackburn, editors, *PLDI'15*, pages 1–11. ACM, 2015.
15. S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part i: Faithful rounding. *SIAM Journal on Scientific Computing*, 31(1), 2008.
16. Alexey Solovyev, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. Rigorous estimation of floating-point round-off errors with symbolic Taylor expansions. In Nikolaj Bjørner and Frank de Boer, editors, *FM 2015: Formal Methods*, 2015.
17. Laura Titolo, Marco A. Feliú, Mariano Moscato, and César A. Muñoz. An abstract interpretation framework for the round-off error analysis of floating-point programs. In *VMCAI*, 2018.