# An Abstract Interpretation Framework for the Round-Off Error Analysis of Floating-Point Programs

Laura Titolo[1], Marco A. Feliú[1], Mariano Moscato[1⋆], and César A. Muñoz[2]

[1] National Institute of Aerospace,
{laura.titolo, marco.feliu, mariano.moscato}@nianet.org
[2] NASA Langley Research Center,
cesar.a.munoz@nasa.gov

**Abstract.** This paper presents an abstract interpretation framework for the round-off error analysis of floating-point programs. This framework defines a parametric abstract analysis that computes, for each combination of ideal and floating-point execution path of the program, a sound over-approximation of the accumulated floating-point round-off error that may occur. In addition, a Boolean expression that characterizes the input values leading to the computed error approximation is also computed. An abstraction on the control flow of the program is proposed to mitigate the explosion of the number of elements generated by the analysis. Additionally, a widening operator is defined to ensure the convergence of recursive functions and loops. An instantiation of this framework is implemented in the prototype tool PRECiSA that generates formal proof certificates stating the correctness of the computed round-off errors.

## 1 Introduction

Floating-point numbers are often used as a finite representation of real numbers in computer programs. While floating-point numbers offer a good compromise between efficiency and precision for most applications, round-off errors in floating-point computations may be unacceptably large for some applications. In particular, in safety-critical systems, even small computational errors may have catastrophic consequences when they are not appropriately accounted for. To guarantee the safety of such systems, it is essential to correctly characterize the difference between a computed result and its ideal real number computation and the impact of this difference in the control-flow of a program.

Significant progress has been made in the last decade in the formal analysis of floating-point computations [5,8,11,15,26,31]. However, as stated in [2], none of the proposed approaches provides at the same time (i) a rigorous round-off errors analysis that generates externally checkable proofs certificates, (ii) the possibility

⋆ Research by the first three authors was supported by the National Aeronautics and Space Administration under NASA/NIA Cooperative Agreement NNL09AA00A.

of handling a wide variety of mathematical operators, and (iii) sound support for typical programming language constructs such as conditionals, recursion, and loops. Another feature, which is not supported by the current errors analysis tools, is *compositionality*, i.e., the ability of analyzing a program in a modular way. This property is essential for obtaining a scalable and efficient approach.

This paper presents an abstract interpretation framework for the round-off error analysis of floating-point programs that addresses all the concerns above. The proposed framework defines a parametric semantics that collects, for each combination of ideal and floating-point computational path of a functional program, an error expression representing a provably sound upper-bound of the accumulated round-off error. Intuitively, the semantics associates conditions to each computed round-off error. The information accumulated in these conditions includes the path conditions, domain conditions ensuring that all expressions are total, e.g., divisors are non-zero, and additional conditions that enable tighter round-off errors for particular values. These conditions not only allow for more precise estimations of the round-off errors, but also enable the characterization of the input values that may lead to errors larger than expected.

The defined semantics is parametric with respect to round-off error bounds defined for a set of arithmetic operators. Hence, the analysis supports the extension of the programming language with new built-in operators as long as sound upper bounds of the operators' round-off errors are provided. The semantics is also parametric with respect to a set of execution paths of interests. These paths are individually examined by the analysis, while the other paths are condensed together in a sole abstract execution path. This abstraction makes the analysis more efficient and enables the analysis of programs with several nested conditionals. Finally, the semantics is parametric with respect to abstract domains of the real and Boolean expressions. Hence, the analysis supports different rigorous enclosure methods such as interval arithmetic, affine arithmetic, Bernstein and Taylor models, etc.

An instance of the presented framework has been implemented in the prototype tool PRECiSA. The input to PRECiSA is a functional program consisting of a set of floating-point functions. The output is a set of round-off error bounds with their associated conditions. Numerical values for these expressions are computed using an optimizer based on a formally verified branch-and-bound algorithm. PRECiSA generates proof certificates in the form of lemmas stating an accumulated round-off error estimation for each function in the program. These lemmas are equipped with proof scripts that automatically discharge them in an interactive theorem prover.

The paper is organized as follows. In Section 2, a formalization of floating-point round-off errors is presented. Section 3 presents the concrete semantics that computes the set of conditional error expressions associated to a program. In Section 4, the abstraction scheme and the widening operator are defined. A prototype tool that implements an instance of the proposed framework is presented in Section 5. Related work is discussed in Section 6. Section 7 concludes the paper.

## 2 Formalization of Floating-Point Round-off Errors

A floating-point number can be formalized as a pair of integers $(m, e) \in \mathbb{Z}^2$ [3,10] where $m$ is called the *significand* and $e$ the *exponent* of the float. A floating-point *format* $f$ is defined as a pair of integers $(p, e_{min})$, where $p$ is called the *precision* and $e_{min}$ is called the *minimal exponent*. Given a base $\beta$, a pair $(m, e) \in \mathbb{Z}^2$ represents a floating-point number in the format $(p, e_{min})$ if and only if it holds that $|m| < \beta^p$ and $-e_{min} \leq e$. For instance, IEEE single and double precision floating-point numbers are specified by the formats $(24, 149)$ and $(53, 1074)$, respectively.

A conversion function $R : \mathbb{Z}^2 \to \mathbb{R}$ is defined to refer to the real number represented by a given float, i.e., $R((m, e)) = m \cdot \beta^e$. Since the function $R$ is not injective, the representation of floating-point numbers is redundant. Therefore, notions about normality and canonicity are needed. A *canonical* float is a float such that is either a normal or subnormal. A *normal* float is a float such that the significand cannot be multiplied by the radix and still fit in the format. A *subnormal* is a float having the minimal exponent such that its significand can be multiplied by the radix and still fit in the format. Henceforth, $\mathbb{F}$ represents the set of floating-point numbers in canonical form. The expression $\tilde{v}$ will denote a floating-point number $(m, e)$ in $\mathbb{F}$.

The expression $F_f(r)$ denotes the floating-point number in format $f$ *closest* to $r$. The format $f$ will be omitted when clear from the context. Let $\tilde{v}$ be a floating-point number that represents a real number $r$, the difference $|R(\tilde{v}) - r|$ is called the *round-off error* (or *rounding error*) of $\tilde{v}$ with respect to $r$. The *unit in the last place* (*ulp*) is a measure of the precision of a floating-point number as a representation of a real number. Given $r \in \mathbb{R}$, $ulp(r)$ represents the difference between two closest consecutive floating-point numbers $\tilde{v}_1$ and $\tilde{v}_2$ such that $\tilde{v}_1 \leq r \leq \tilde{v}_2$ and $\tilde{v}_1 \neq \tilde{v}_2$. It is defined in [3] as $ulp(\tilde{v}) = \beta^{e_{\tilde{v}}}$, where $e_{\tilde{v}}$ is the exponent of the *canonical* form of $\tilde{v}$ that is the floating-point number closest to $r$. The *ulp* can be used to bound the round-off error of a real number $r$ with respect to its floating-point representation in the following way:

$$|R(F(r)) - r| \leq \tfrac{1}{2}\, ulp(r). \tag{2.1}$$

Given a set $\widetilde{\Omega}$ of pre-defined arithmetic floating-point operations, the corresponding set $\Omega$ of operations over real numbers, a denumerable set $\mathbb{V}$ of variables representing real values, and a denumerable set $\widetilde{\mathbb{V}}$ of variables representing floating-point values, where $\mathbb{V}$ and $\widetilde{\mathbb{V}}$ are disjoint, the sets $\mathbb{A}$ and $\widetilde{\mathbb{A}}$ of arithmetic expressions over real numbers and over floating-point numbers, respectively, are defined by the following grammar.

$$A ::= d \mid x \mid op(A, \ldots, A) \qquad\qquad \widetilde{A} ::= \tilde{d} \mid \tilde{x} \mid \widetilde{op}(\widetilde{A}, \ldots, \widetilde{A})$$

where $A \in \mathbb{A}$, $d \in \mathbb{R}$, $x \in \mathbb{V}$, $op \in \Omega$, $\widetilde{A} \in \widetilde{\mathbb{A}}$, $\tilde{d} \in \mathbb{F}$, $\tilde{x} \in \widetilde{\mathbb{V}}$, and $\widetilde{op} \in \widetilde{\Omega}$. It is assumed that there is a function $\chi_r : \widetilde{\mathbb{V}} \to \mathbb{V}$ that associates to each floating-point variable $\tilde{x}$ a variable $x \in \mathbb{V}$ representing the real value of $\tilde{x}$. Given a variable assignment $\sigma$ :

$\mathbb{V} \to \mathbb{R}$, $eval_{\mathbb{A}}(\sigma, A) \in \mathbb{R}$ denotes the evaluation of the real arithmetic expression $A$ with respect to $\sigma$. Similarly, given $\tilde{A} \in \widetilde{\mathbb{A}}$ and $\tilde{\sigma} : \widetilde{\mathbb{V}} \to \mathbb{F}$, $\widetilde{eval}_{\widetilde{\mathbb{A}}}(\tilde{\sigma}, \tilde{A}) \in \mathbb{F}$ denotes the evaluation of the floating-point arithmetic expression $\tilde{A}$ with respect to $\tilde{\sigma}$. The (partial) order relation between arithmetic expressions is defined as follows: $A_1 \leq A_2$ if and only if for all $\sigma : \mathbb{V} \to \mathbb{R}$, $eval_{\mathbb{A}}(\sigma, A_1) \leq eval_{\mathbb{A}}(\sigma, A_2)$.

The round-off error of the floating-point expression $\widetilde{op}(\tilde{v}_1, \ldots, \tilde{v}_n)$ with respect to the real-valued expression $op(r_1, \ldots, r_n)$, where $\widetilde{op}$ is a floating-point operator representing a real-valued operator $op$ and $\tilde{v}_i$ is a floating-point value representing a real value $r_i$, for $1 \leq i \leq n$, depends of (a) the error introduced by the application of $\widetilde{op}$ versus $op$ and (b) the propagation of the errors carried out by the arguments, i.e., the difference between $\tilde{v}_i$ and $r_i$, for $1 \leq i \leq n$, in the application. In the case of arithmetic operators, the IEEE-754 standard states that every basic operation is correctly rounded, therefore it should be performed as if it would be calculated with infinite precision and then rounded to the nearest floating-point value. Then, from Formula (2.1), the application of an $n$-ary floating-point operator $\widetilde{op}$ to the floating-point values $\tilde{v}_1, \ldots, \tilde{v}_n$ must fulfill the following condition.

$$|R(\widetilde{op}(\tilde{v}_i)_{i=1}^n) - op(R(\tilde{v}_i))_{i=1}^n| \leq \tfrac{1}{2}\, ulp(op(R(\tilde{v}_i))_{i=1}^n), \tag{2.2}$$

where the notation $f(x_i)_{i=1}^n$ is used to represent $f(x_1, \ldots, x_n)$.

To estimate how the errors of the arguments are propagated to the result of the application of the operator, it is necessary to bound the difference between the application of the real operator on real values and the application of the same operator on the floating-point arguments. The expression $\epsilon_{op}(e_i)_{i=1}^n$ is used to represent such difference, where each $e_i$ is a bound of the round-off error carried by every floating-point $\tilde{v}_i$ representing a real value $r_i$, i.e., $|R(\tilde{v}_i) - r_i| \leq e_i$. Therefore, $\epsilon_{op}(e_i)_{i=1}^n$ satisfies the following condition.

$$|op(R(\tilde{v}_i))_{i=1}^n - op(r_i)_{i=1}^n| \leq \epsilon_{op}(e_i)_{i=1}^n. \tag{2.3}$$

The following bound of the round-off error between the floating-point expression and the real-valued counterpart follows from Formula (2.2), Formula (2.3), and the triangle inequality.

$$|R(\widetilde{op}(\tilde{v}_i)_{i=1}^n) - op(r_i)_{i=1}^n| \leq \epsilon_{op}(e_i)_{i=1}^n + \tfrac{1}{2}\, ulp(op(R(\tilde{v}_i))_{i=1}^n). \tag{2.4}$$

In this paper, for a given expression, the round-off error in the right-hand side of Formula (2.4) is expressed as an error expression.

**Definition 1 (Error Expression).** *An* error expression *is an arithmetic expression or the element $+\infty$ representing an arbitrary large round-off error.*

The domain of error expressions is denoted as $\mathbb{E}$ and it is defined as $\mathbb{E} := \mathbb{A} \cup \{+\infty\}$. The order relation on error expressions naturally extends the one on arithmetic expressions by stating that for all $e \in \mathbb{E}$, $e \leq +\infty$. The function *max* (respectively *min*) returns the maximum (respectively minimum) of a set error expressions with respect to the order relation $\leq$. The tuple $(\mathbb{E}, \leq, max, min, +\infty, 0)$ is a

complete lattice, where *max* is the least upper bound, *min* is the greatest lower bound, $+\infty$ is the greatest element of the domain, and 0 is the least element of the domain.

Additional conditions are needed in Formula (2.4) when the operators are not total. For example, when dealing with the division operation, it is necessary to guarantee that the second argument of both the floating-point operator and the real-valued operator is not zero. Furthermore, some arithmetic operations are associated to tighter error bounds under certain conditions. These conditions can be used to refine the estimation of the round-off error. Boolean expressions are used to model such conditions.

The sets $\mathbb{B}$ and $\widetilde{\mathbb{B}}$ of Boolean expressions over real numbers and over floating-point numbers, respectively, are defined by the following grammar.

$$B ::= \mathit{true} \mid \mathit{false} \mid B \wedge B \mid B \vee B \mid \neg B \mid A < A \mid A = A$$
$$\widetilde{B} ::= \mathit{true} \mid \mathit{false} \mid \widetilde{B} \wedge \widetilde{B} \mid \widetilde{B} \vee \widetilde{B} \mid \neg \widetilde{B} \mid \widetilde{A} < \widetilde{A} \mid \widetilde{A} = \widetilde{A}$$

where $B \in \mathbb{B}$, $A \in \mathbb{A}$, $\widetilde{B} \in \widetilde{\mathbb{B}}$, and $\widetilde{A} \in \widetilde{\mathbb{A}}$. The conjunction $\wedge$, disjunction $\vee$, negation $\neg$, *true*, and *false* have the usual classical logic meaning.

Given a variable assignment $\sigma : \mathbb{V} \to \mathbb{R}$, $\mathit{eval}_{\mathbb{B}}(\sigma, B) \in \{\mathit{true}, \mathit{false}\}$ denotes the evaluation of the real Boolean expression $B$. In the same way, given $\widetilde{B} \in \widetilde{\mathbb{B}}$ and $\widetilde{\sigma} : \widetilde{\mathbb{V}} \to \mathbb{F}$, $\widetilde{\mathit{eval}}_{\widetilde{\mathbb{B}}}(\widetilde{\sigma}, \widetilde{B}) \in \{\mathit{true}, \mathit{false}\}$ denotes the evaluation of the floating-point Boolean expression $\widetilde{B}$. The (partial) order relation between Boolean expressions over real numbers is defined as follows: $B_1 \Rightarrow B_2$ if and only if for all $\sigma : \mathbb{V} \to \{\mathit{true}, \mathit{false}\}$, $\mathit{eval}_{\mathbb{B}}(\sigma, B_1)$ implies $\mathit{eval}_{\mathbb{B}}(\sigma, B_2)$. Similarly, for floating-point Boolean expressions, the order relation is defined as follows: $\widetilde{B}_1 \Rightarrow \widetilde{B}_2$ if and only if for all $\tilde{\sigma} : \widetilde{\mathbb{V}} \to \{\mathit{true}, \mathit{false}\}$, $\widetilde{\mathit{eval}}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \widetilde{B}_1)$ implies $\widetilde{\mathit{eval}}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \widetilde{B}_2)$. The symbol *true* (respectively *false*) is the greatest (respectively least) Boolean expression of both domains $\mathbb{B}$ and $\widetilde{\mathbb{B}}$. The equivalence relation derived from $\Rightarrow$ is defined as $B_1 \Leftrightarrow B_2$ if and only if $B_1 \Rightarrow B_2$ and $B_2 \Rightarrow B_1$. In the following, by abuse of notation, a formula $B \in \mathbb{B} \cup \widetilde{\mathbb{B}}$ and its equivalence class will be denoted with the same symbol.

The function $R_{\mathbb{B}} : \widetilde{\mathbb{B}} \to \mathbb{B}$ that converts a Boolean expression on floating-point numbers to a Boolean expression on real numbers is defined by simply replacing each floating-point operation with the corresponding operation on real numbers and by applying $R$ and $\chi_r$ to floating-point values and variables, respectively.

Henceforth, it is assumed that for any floating-point operator of interest *op* there exists at least one formula of the following form that holds for all $e_1, \ldots, e_n \in \mathbb{E}$ such that $|R(\tilde{v}_i) - r_i| \leq e_i$ with $1 \leq i \leq n$,

$$\phi_{op}(r_i)_{i=1}^n \wedge \phi_{\widetilde{op}}(\tilde{v}_i)_{i=1}^n \text{ implies } |R(\widetilde{op}(\tilde{v}_i)_{i=1}^n) - op(r_i)_{i=1}^n| \leq \epsilon_{\widetilde{op}}(r_i, e_i)_{i=1}^n, \quad (2.5)$$

where $\phi_{op}(r_i)_{i=1}^n \in \mathbb{B}$, $\phi_{op}(r_i)_{i=1}^n \not\Rightarrow \mathit{false}$, $\phi_{\widetilde{op}}(\tilde{v}_i)_{i=1}^n \in \widetilde{\mathbb{B}}$, $\phi_{\widetilde{op}}(\tilde{v}_i)_{i=1}^n \not\Rightarrow \mathit{false}$, and $\epsilon_{\widetilde{op}} : \mathbb{A}^n \times \mathbb{E}^n \to \mathbb{E}$. For the same floating-point operator there may be more than one formula of the form of Formula (2.5). In this case, the disjunction of all conditions in the left-hand side of Formula (2.5) should be complete for the domain of the operator. The framework presented in this paper does not require

those conditions to be disjoint, but better estimations are usually computed when these conditions are disjoint.

*Example 1.* Instances of Formula (2.5) for the four basic arithmetic operators are defined below.

- $\epsilon_{\tilde{+}}(r_1, e_1, r_2, e_2) := e_1 + e_2 + 1/2\, ulp(|r_1 + r_2| + e_1 + e_2)$, $\phi_{+}(r_1, r_2) := true$, and $\phi_{\tilde{+}}(\tilde{v}_1, \tilde{v}_2) := true$.
- $\epsilon_{\tilde{-}}(r_1, e_1, r_2, e_2) := e_1 + e_2 + 1/2\, ulp(|r_1 - r_2| + e_1 + e_2)$, $\phi_{-}(r_1, r_2) := true$, and $\phi_{\tilde{-}}(\tilde{v}_1, \tilde{v}_2) := \tilde{v}_2 \tilde{/} 2 > \tilde{v}_1 \vee \tilde{v}_1 > 2 \tilde{*} \tilde{v}_2$.
- $\epsilon_{\tilde{-}}(r_1, e_1, r_2, e_2) := e_1 + e_2$, $\phi_{-}(r_1, r_2) := true$ and $\phi_{\tilde{-}}(\tilde{v}_1, \tilde{v}_2) := \tilde{v}_2 \tilde{/} 2 \leq \tilde{v}_1 \wedge \tilde{v}_1 \leq 2 \tilde{*} \tilde{v}_2$.
- $\epsilon_{\tilde{*}}(r_1, e_1, r_2, e_2) := |r_1| e_2 + |r_2| e_1 + e_1 e_2 + 1/2\, ulp((|r_1| + e_1)(|r_2| + e_2))$, $\phi_{*}(r_1, r_2) := true$, and $\phi_{\tilde{*}}(\tilde{v}_1, \tilde{v}_2) := true$.
- $\epsilon_{\tilde{/}}(r_1, e_1, r_2, e_2) := \frac{|r_1| e_2 + |r_2| e_1}{r_2 r_2 - e_2 |r_2|} + 1/2\, ulp(\frac{|r_1| + e_1}{|r_2| - e_2})$, $\phi_{/}(r_1, r_2) := r_2 \neq 0$, and $\phi_{\tilde{/}}(\tilde{v}_1, \tilde{v}_2) := \tilde{v}_2 \neq 0$.

For instance, the round-off error of the sum includes the propagation of the errors of the operands ($e_1$ and $e_2$) and the error of rounding the result of the sum ($1/2\, ulp(|r_1 - r_2| + e_1 + e_2)$). In the case of the division operator, Boolean conditions are used to guarantee the validity of the operation, i.e., the conditions $\phi_{/}$ and $\phi_{\tilde{/}}$ state that the divisors of the real and floating point expressions, respectively, are different from zero. In the case of the subtraction operator, conditions that improve the error approximation are provided. Indeed, in [14], it is proven that the floating-point subtraction $x \tilde{-} y$ is computed exactly when $y \tilde{/} 2 \leq x \leq 2 \tilde{*} y$.

## 3 Concrete Denotational Semantics

This section presents a compositional structural denotational semantics for a generic declarative programming language. This semantics collects information about the round-off error of floating point operations and relies on the floating-point error formalization presented in Section 2. This semantics is an enhancement of the one introduced in [26] and it uses a more expressive domain.

The expression language considered in this paper contains conditionals, let expressions, and function calls, possibly recursive. Given a set $\widetilde{\Omega}$ of pre-defined arithmetic floating-point operations, a set $\Sigma$ of function symbols, and a denumerable set $\widetilde{\mathbb{V}}$ of floating-point variables, $\mathbb{S}$ denotes the set of program expressions. The syntax of programs in $\mathbb{S}$ is given by the following grammar, where the syntax of floating-point arithmetic expressions given in Section 2 is augmented with a function call.

$$\widetilde{A} ::= \tilde{d} \mid F(d) \mid \tilde{x} \mid \widetilde{op}(\widetilde{A}, \ldots, \widetilde{A}) \mid \ f(\widetilde{A}, \ldots, \widetilde{A})$$
$$S ::= \widetilde{A} \mid \textit{if } \widetilde{B} \textit{ then } S \textit{ else } S \mid \textit{let } \tilde{x} = \widetilde{A} \textit{ in } S$$

where $\widetilde{A} \in \widetilde{\mathbb{A}}$, $\widetilde{B} \in \widetilde{\mathbb{B}}$, $\tilde{d} \in \mathbb{F}$, $d \in \mathbb{R}$, $\tilde{x} \in \widetilde{\mathbb{V}}$, $\widetilde{op} \in \widetilde{\Omega}$, and $f \in \Sigma$. Bounded recursion is added to the language as syntactic sugar using the convention $for(i, j, S, g) := if\ i > j\ then\ S\ else\ g(j, for(i, j - 1, S, g))$.

A program is defined as a set of *function declarations* of the form $f(\tilde{x}_1, \ldots, \tilde{x}_n) = S$, where $\tilde{x}_1, \ldots, \tilde{x}_n$ are pairwise distinct variables in $\widetilde{\mathbb{V}}$ and all free variables appearing in $S$ are in $\{\tilde{x}_1, \ldots, \tilde{x}_n\}$. The natural number $n$ is called the *arity* of $f$. Henceforth, it is assumed that programs are well-formed in the sense that for every function call $f(\tilde{x}_1, \ldots, \tilde{x}_n)$ that occurs in a program $P$, a unique function $f$ of arity $n$ is defined in $P$. The set of programs is denoted as $\mathbb{P}$.

The proposed semantics collects, for each program path, the corresponding path conditions (for both the real and the floating-point execution), and two expressions representing (1) the value of the output assuming the use of real arithmetic and (2) an upper bound for the accumulated round-off error that might affect the result due to floating-point operations. Since the semantics collects information about real and floating-point execution paths, it is possible to consider the error of taking the incorrect branch compared to the ideal execution using real arithmetic. This enables a sound treatment of unstable tests.

**Definition 2 (Test Stability).** *A conditional statement if $\tilde{\phi}$ then $\tilde{E}_1$ else $\tilde{E}_2$ is said to be* unstable *if there exist two assignments $\tilde{\sigma} : \widetilde{\mathbb{V}} \to \mathbb{F}$ and $\sigma : \mathbb{V} \to \mathbb{R}$ such that for all $\tilde{x} \in \widetilde{\mathbb{V}}$, $\sigma(\chi_r(\tilde{x})) = R(\tilde{\sigma}(\tilde{x}))$ and $eval_{\mathbb{B}}(\sigma, R_{\mathbb{B}}(\tilde{\phi})) \neq \widetilde{eval}_{\widetilde{\mathbb{B}}}(\tilde{\sigma}, \tilde{\phi})$. Otherwise the conditional expression is said to be* stable.

In other words, a conditional statement is unstable when there exists an assignment from the variables in $\tilde{\phi}$ to $\mathbb{F}$ such that $\tilde{\phi}$ and $R_{\mathbb{B}}(\tilde{\phi})$ evaluate to different Boolean values.

A *condition* is a set of pairs of the form $(\phi, \tilde{\phi})$, with $\phi \in \mathbb{B}$ and $\tilde{\phi} \in \widetilde{\mathbb{B}}$. The domain of conditions is $(\wp(\mathbb{B} \times \widetilde{\mathbb{B}}), \hat{\Rightarrow}, \hat{\vee}, \hat{\wedge}, \{(true, true)\}, \{(false, false)\})$, where

- $\hat{\Rightarrow}$ is the order relation over $\wp(\mathbb{B} \times \widetilde{\mathbb{B}})$ defined as for all $\eta_1, \eta_2 \in \wp(\mathbb{B} \times \widetilde{\mathbb{B}})$, $\eta_1 \hat{\Rightarrow} \eta_2$ if and only if $\bigvee_{(b_1, \tilde{b}_1) \in \eta_1} (b_1 \wedge \tilde{b}_1) \Rightarrow \bigvee_{(b_2, \tilde{b}_2) \in \eta_2} (b_2 \wedge \tilde{b}_2)$,
- the equivalence relation $\hat{\Leftrightarrow}$ derived from $\hat{\Rightarrow}$ is defined as follows, $\eta_1 \hat{\Leftrightarrow} \eta_2$ if and only if $\eta_1 \hat{\Rightarrow} \eta_2$ and $\eta_2 \hat{\Rightarrow} \eta_1$, and the equivalence class of a condition $\eta$ is denoted as $[\eta]_{\hat{\Leftrightarrow}}$,
- $\hat{\vee}$ is the least upper bound defined as $\eta_1 \hat{\vee} \eta_2 = [\eta_1 \cup \eta_2]_{\hat{\Leftrightarrow}}$,
- $\hat{\wedge}$ is the greatest lower bound defined as $\eta_1 \hat{\wedge} \eta_2 = \bigcup\{(b_1 \wedge b_2, \tilde{b}_1 \wedge \tilde{b}_2) \mid (b_1, \tilde{b}_1) \in \eta_1 (b_2, \tilde{b}_2) \in \eta_2\}$,
- $\{(true, true)\}$ is the greatest element of the domain, and
- $\{(false, false)\}$ is the least element of the domain.

Paths in the control flow of a program are represented by sequences, possibly empty, of 0's and 1's.

**Definition 3 (Decision path).** *A* decision path $\pi$ *is defined by the grammar $\pi = \varepsilon \mid \pi \cdot 0 \mid \pi \cdot 1$, where $\varepsilon$ denotes the empty path and $\cdot$ is the concatenation operator.*

The domain of all decision paths is denoted by *Path*. A decision path $\pi$ models all the decision paths $\pi'$ such that $\pi$ is prefix of $\pi'$. Given $\pi_1, \pi_2 \in Path$, the order relation on decision paths is defined as $\pi_1 \leq_{prefix} \pi_2$ if and only if

$\pi_1$ is a prefix of $\pi_2$. A decision path univocally identifies a subprogram or subexpression inside the input program. Subexpressions corresponding to the *then* branch of a conditional statement are identified by the index 1. Conversely, the subexpressions corresponding to the *else* branch are identified by the index 0. For example, consider the following program expression:

$$E = \textit{if } \tilde{x} > 0 \textit{ then } (\textit{if } \tilde{y} > 2 \textit{ then } 5 \textit{ else } \tilde{y} + 1) \textit{ else } (\textit{if } \tilde{z} > 0 \textit{ then } \tilde{x} + \tilde{z} \textit{ else } \tilde{y} * \tilde{z})$$

All the decision paths of expression $E$ are identified by $\varepsilon$. The path corresponding to the arithmetic expression $\tilde{y}+1$ is $1{\cdot}0$, and the path corresponding to expression $\tilde{x} + \tilde{z}$ is $0 \cdot 1$.

The semantics collects information in the form of *conditional error bounds*.

**Definition 4 (Conditional Error Bound).** *A* conditional error bound *is an expression of the form* $\langle \eta \rangle_t \twoheadrightarrow (r, e)^\pi$*, where* $\eta \in \wp(\mathbb{B} \times \widetilde{\mathbb{B}})$*,* $r \in \mathbb{A}$*,* $e \in \mathbb{E}$*,* $\pi \in \mathit{Path}$*, and* $t \in \{\mathbf{s}, \mathbf{u}\}$*. A conditional error bound is said to be* valid *if it exists* $(\phi, \tilde{\phi}) \in \eta$*,* $\phi \not\Rightarrow \mathit{false}$ *and* $\tilde{\phi} \not\Rightarrow \mathit{false}$*.*

Intuitively, $\langle \eta \rangle_t \twoheadrightarrow (r, e)^\pi$ indicates that for the decision path $\pi$, if the condition $\eta$ is satisfied, the output of the ideal real numbers implementation of the program is $r$ and the round-off error of the floating-point implementation is bounded by $e$. The sub-index $t$ is used to mark by construction whether a conditional error bound is unstable ($t = \mathbf{u}$), or stable ($t = \mathbf{s}$).

Conditional error bounds are ordered in the following way $\langle \eta_1 \rangle_{t_1} \twoheadrightarrow (r_1, e_1)^{\pi_1} \leq \langle \eta_2 \rangle_{t_2} \twoheadrightarrow (r_2, e_2)^{\pi_2}$ if and only if $\eta_1 \hat{\Rightarrow} \eta_2$, $r_1 = r_2$, $e_1 \leq e_2$, $\pi_2 \leq_{\mathit{prefix}} \pi_1$, and $t_1 = t_2$. The domain $\mathbf{C}$ of conditional error bounds is defined as a set of tuples in $\wp(\mathbb{B} \times \widetilde{\mathbb{B}}) \times \mathbb{A} \times \mathbb{E} \times \mathit{Path} \times \{\mathbf{s}, \mathbf{u}\}$. Sets of conditional error bounds are (partially) ordered as follows. For all $C_1, C_2 \subseteq \mathbf{C}$, $C_1 \sqsubseteq C_2$ if and only iff for all $c_1 \in C_1$, there exists $c_2 \in C_2$ such that $c_1 \leq c_2$. The equivalence relation derived from $\sqsubseteq$ is defined as $C_1 \equiv C_2$ if and only if $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. In the following, by abuse of notation, the quotient of $\sqsubseteq$ over equivalence classes will be denoted with the same symbol. Furthermore, sets of conditional error bounds will be used modulo $\equiv$ and their class will be denoted as $\mathbb{C}$. The domain $(\mathbb{C}, \sqsubseteq, \bigsqcup, \bigsqcap, [\mathbf{C}]_\equiv, \varnothing)$ is a complete lattice where the least upper bound is defined as $C_1 \sqcup C_2 \coloneqq [C_1 \cup C_2]_\equiv$ and the greatest lower bound is defined as $C_1 \sqcap C_2 \coloneqq [\{c \in \mathbf{C} \mid \exists c_1 \in C_1 . c \leq c_1, \exists c_2 \in C_2 . c \leq c_2\}]_\equiv$.

An *environment* is defined as a function mapping a variable to a set of conditional error bounds, i.e., $\mathit{Env} = \widetilde{\mathbb{V}} \to \mathbb{C}$. The empty environment is denoted as $\bot_{\mathit{Env}}$ and maps every variable to the empty set $\varnothing$. Let $\mathbb{M} \coloneqq \{f(\tilde{x}_1, \ldots, \tilde{x}_n) \mid f \in \Sigma, \tilde{x}_1, \ldots, \tilde{x}_n \in \widetilde{\mathbb{V}}\}$ be the set of all possible function calls. An *interpretation* is a function $I : \mathbb{M} \to \mathbb{C}$ modulo variance[3]. The set of all interpretations is denoted as $\mathbb{I}$. The empty interpretation is denoted as $\bot_{\mathbb{I}}$ and maps everything to $\varnothing$.

Let $\widetilde{op}$ be an $n$-ary floating-point operator in $\widetilde{\Omega}$ such that $op$ in $\Omega$ is its real-valued counterpart and there exist $\epsilon_{\widetilde{op}} : \mathbb{A}^n \times \mathbb{E}^n \to \mathbb{E}$, $\phi_{op}(r_i)_{i=1}^n \in \mathbb{B}$ and

[3] Two functions $I_1, I_2 : \mathbb{M} \to \mathbb{C}$ are variants if for each $m \in \mathbb{M}$ there exists a renaming $\rho$ such that $(I_1(m))\rho = I_2(m\rho)$.

$\phi_{\widetilde{op}}(\tilde{v}_i)_{i=1}^n \in \widetilde{\mathbb{B}}$ such that Formula (2.5) holds. Given $\sigma \in Env$ and $I \in \mathbb{I}$, the semantics of program expressions, $\mathcal{E} : \mathbb{S} \times Env \times \mathbb{I} \times Path \to \mathbb{C}$, returns the set of conditional error bounds representing an upper bound of the round-off error for each execution path, together with the corresponding conditions. The function $\chi_e : \widetilde{\mathbb{V}} \to \mathbb{V}$ associates to each floating-point variable $\tilde{x}$ a variable in $\mathbb{V}$ representing the error of $\tilde{x}$. In the following, for the sake of simplicity, the singleton condition $\langle\{(\phi, \tilde{\phi})\}\rangle$ will be denoted as $\langle\phi, \tilde{\phi}\rangle$.

$$\mathcal{E}[\![\tilde{d}]\!]^{\pi}_{(\sigma,I)} \coloneqq \{\langle true, true\rangle_{\mathbf{s}} \twoheadrightarrow (R(\tilde{d}), 0)^{\pi}\}$$

$$\mathcal{E}[\![F(d)]\!]^{\pi}_{(\sigma,I)} \coloneqq \{\langle true, true\rangle_{\mathbf{s}} \twoheadrightarrow (d, |d - F(d)|)^{\pi}\}$$

$$\mathcal{E}[\![\tilde{x}]\!]^{\pi}_{(\sigma,I)} \coloneqq \begin{cases} \{\langle true, true\rangle_{\mathbf{s}} \twoheadrightarrow (\chi_r(\tilde{x}), \chi_e(\tilde{x}))^{\pi}\} & \text{if } \sigma(\tilde{x}) = \varnothing \\ \sigma(\tilde{x}) & \text{otherwise} \end{cases}$$

$$\mathcal{E}[\![\widetilde{op}(\widetilde{A}_i)_{i=1}^n]\!]^{\pi}_{(\sigma,I)} \coloneqq$$

$$\bigsqcup\{\langle \bigwedge_{i=1}^n \phi_i \wedge \phi_{op}(r_i)_{i=1}^n, \bigwedge_{i=1}^n \tilde{\phi}_i \wedge \phi_{\widetilde{op}}(\widetilde{A}_i)_{i=1}^n\rangle_{\mathbf{s}} \twoheadrightarrow (op(r_i)_{i=1}^n, \epsilon_{\widetilde{op}}(r_i, e_i)_{i=1}^n)^{\pi} \mid \forall 1 \le i \le n:$$

$$\langle\phi_i, \tilde{\phi}_i\rangle_{\mathbf{s}} \twoheadrightarrow (r_i, e_i)^{\pi_i} \in \mathcal{E}[\![\widetilde{A}_i]\!]^{\pi}_{(\sigma,I)}, \bigwedge_{i=1}^n \phi_i \wedge \phi_{op}(r_i)_{i=1}^n \not\Rightarrow false, \bigwedge_{i=1}^n \tilde{\phi}_i \wedge \phi_{\widetilde{op}}(\widetilde{A}_i)_{i=1}^n \not\Rightarrow false\}$$

$$\mathcal{E}[\![let\ \tilde{x} = \widetilde{A}\ in\ S]\!]^{\pi}_{(\sigma,I)} \coloneqq \mathcal{E}[\![S]\!]^{\pi}_{(\sigma[\tilde{x} \mapsto \mathcal{E}[\![\widetilde{A}]\!]^{\pi}_{(\sigma,I)}],I)}$$

$$\mathcal{E}[\![if\ \widetilde{B}\ then\ S_1\ else\ S_2]\!]^{\pi}_{(\sigma,I)} \coloneqq \mathcal{E}[\![S_1]\!]^{\pi\cdot 1}_{(\sigma,I)} \Downarrow_{(R_{\mathbb{B}}(\widetilde{B}),\widetilde{B})} \sqcup\ \mathcal{E}[\![S_2]\!]^{\pi\cdot 0}_{(\sigma,I)} \Downarrow_{(\neg R_{\mathbb{B}}(\widetilde{B}),\neg\widetilde{B})} \sqcup$$

$$\bigsqcup\{\langle\phi_2, \tilde{\phi}_1\rangle_{\mathbf{u}} \twoheadrightarrow (r_2, e_1 + |r_1 - r_2|)^{\varepsilon} \mid \langle\phi_1, \tilde{\phi}_1\rangle_{t_1} \twoheadrightarrow (r_1, e_1)^{\pi_1} \in \mathcal{E}[\![S_1]\!]^{\pi\cdot 0}_{(\sigma,I)},$$
$$\langle\phi_2, \tilde{\phi}_2\rangle_{t_2} \twoheadrightarrow (r_2, e_2)^{\pi_2} \in \mathcal{E}[\![S_2]\!]^{\pi\cdot 1}_{(\sigma,I)}\} \Downarrow_{(\neg R_{\mathbb{B}}(\widetilde{B}),\widetilde{B})} \sqcup$$

$$\bigsqcup\{\langle\phi_1, \tilde{\phi}_2\rangle_{\mathbf{u}} \twoheadrightarrow (r_1, e_2 + |r_1 - r_2|)^{\varepsilon} \mid \langle\phi_1, \tilde{\phi}_1\rangle_{t_1} \twoheadrightarrow (r_1, e_1)^{\pi_1} \in \mathcal{E}[\![S_1]\!]^{\pi\cdot 1}_{(\sigma,I)},$$
$$\langle\phi_2, \tilde{\phi}_2\rangle_{t_2} \twoheadrightarrow (r_2, e_2)^{\pi_2} \in \mathcal{E}[\![S_2]\!]^{\pi\cdot 0}_{(\sigma,I)}\} \Downarrow_{(R_{\mathbb{B}}(\widetilde{B}),\neg\widetilde{B})}$$

$$\mathcal{E}[\![f(\widetilde{A}_i)_{i=1}^n]\!]^{\pi}_{(\sigma,I)} \coloneqq \bigsqcup\{\langle\phi' \wedge \bigwedge_{i=1}^n \phi_i, \tilde{\phi}' \wedge \bigwedge_{i=1}^n \tilde{\phi}_i\rangle_t \twoheadrightarrow (r', e')^{\pi'} \mid$$

$$\langle\phi, \tilde{\phi}\rangle_t \twoheadrightarrow (r, e)^{\pi'} \in I(f(\tilde{x}_i)_{i=1}^n), \forall 1 \le i \le n: \langle\phi_i, \tilde{\phi}_i\rangle_{t_i} \twoheadrightarrow (r_i, e_i)^{\pi_i} \in \mathcal{E}[\![\widetilde{A}_i]\!]^{\pi}_{(\sigma,I)},$$
$$r' = r[\chi_r(\tilde{x}_i)/r_i]_{i=1}^n,\ e' = e[\chi_e(\tilde{x}_i)/e_i]_{i=1}^n, \phi' = \phi[\chi_r(\tilde{x}_i)/r_i, \chi_e(\tilde{x}_i)/e_i]_{i=1}^n,$$
$$\tilde{\phi}' = \tilde{\phi}[\chi_r(\tilde{x}_i)/r_i, \chi_e(\tilde{x}_i)/e_i]_{i=1}^n,\ \phi' \wedge \bigwedge_{i=1}^n \phi_i \not\Rightarrow false, \tilde{\phi}' \wedge \bigwedge_{i=1}^n \tilde{\phi}_i \not\Rightarrow false\}$$

The semantics of a variable $\tilde{x} \in \widetilde{\mathbb{V}}$ consists of two cases. If $\tilde{x}$ belongs to the environment, then the variable has been previously bound to a program expression $S$ through a let-expression. In this case, the semantics of $\tilde{x}$ is exactly the semantics of $S$. If $\tilde{x}$ does not belong to the environment, then $\tilde{x}$ is a parameter of the function. Here, a new conditional error bound is added with two place holders, $\chi_r(\tilde{x})$ and $\chi_e(\tilde{x})$, representing the real value and the error of $\tilde{x}$, respectively.

The semantics of a floating-point arithmetic operation $\widetilde{op}$ is computed by composing the semantics of its operands. The real value is obtained by applying the correspondent real arithmetic operation $op$ to the real values of the operands, and the new error bound is obtained by applying $\epsilon_{\widetilde{op}}$ to the errors and real values of the operands. The new conditions are obtained as the combination of

the conditions of the operands. Predicates $\phi_{op}$ and $\phi_{\widetilde{op}}$ represent the additional constraints needed when $op$ and $\widetilde{op}$ are not total (as explained in Section 2).

The semantics of the expression *let* $\tilde{x} = \widetilde{A}$ *in* $S$ updates the current environment by associating to variable $\tilde{x}$ the semantics of expression $\widetilde{A}$.

The semantics of the conditional uses an auxiliary operator $\Downarrow$ for propagating new information in the conditions.

**Definition 5 (Condition propagation operator).** *Given* $b \in \mathbb{B}$ *and* $\tilde{b} \in \widetilde{\mathbb{B}}$, $\langle \eta \rangle_t \twoheadrightarrow (r, e)^{\pi} \Downarrow_{(b,\tilde{b})} = \langle \bigcup_{(\phi,\tilde{\phi})\in\eta} (\phi \wedge b, \tilde{\phi} \wedge \tilde{b}) \rangle_t \twoheadrightarrow (r, e)^{\pi}$ *if* $\bigvee_{(\phi,\tilde{\phi})\in\eta} (\phi \wedge b \wedge \tilde{\phi} \wedge \tilde{b}) \not\Rightarrow$ *false, otherwise it is undefined. The definition of* $\Downarrow$ *naturally extends to sets of conditional error bounds: given* $C \in \mathbb{C}$, $C \Downarrow_{(b,\tilde{b})} = \bigcup_{c\in C} c \Downarrow_{(b,\tilde{b})}$.

The semantics of $S_1$ and $S_2$ are enriched with the information about the fact that real and floating-point execution paths match, i.e., both $\widetilde{B}$ and $R_{\mathbb{B}}(\widetilde{B})$ have the same value. If real and floating point execution paths do not coincide, the error of taking one branch instead of the other has to be considered. For example, if $\widetilde{B}$ is satisfied but $R_{\mathbb{B}}(\widetilde{B})$ is not, the *then* branch is taken in the floating point computation, but the *else* would have been taken in the real one. In this case, the error is the difference between the real value of the result of $S_2$ and the floating point result of $S_1$. It has been shown that this error is bounded by the round-off error of $S_1$ plus the difference between the real values of $S_1$ and $S_2$. The condition $(\neg R_{\mathbb{B}}(\widetilde{B}), \widetilde{B})$ is propagated in order to model that $\widetilde{B}$ holds but $R_{\mathbb{B}}(\widetilde{B})$ does not. The conditional error bounds representing this case are marked with $\mathbf{u}$, denoting that the error is due to an unstable test. The parameter $\pi$ of the semantics is augmented by one index that indicates the decision taken: 1 for the *then* and 0 for the *else* branch.

The semantics of a function call combines the conditions coming from the interpretation of the function and the ones coming from the semantics of the parameters. Variables representing real values and errors of formal parameters are replaced with the expressions coming from the semantics of the actual parameters.

The semantics of a program is a function $\mathcal{F} : \mathbb{P} \times Env \to \mathbb{C}$ defined as the least fixed point of the immediate consequence operator $\mathcal{P} : \mathbb{P} \times Env \times \mathbb{I} \to \mathbb{C}$, i.e., given $P \in \mathbb{P}$, $\mathcal{F}[\![P]\!] := lfp(\mathcal{P}[\![P]\!]_{\perp_{\mathbb{I}}})$, which is defined as follows for each function symbol $f$ defined in $P$:

$$\mathcal{P}[\![P]\!]_I(f(\tilde{x}_1 \dots \tilde{x}_n)) := \mathcal{E}[\![S]\!]^{\varepsilon}_{(\perp_{Env}, I)} \text{ if } f(\tilde{x}_1 \dots \tilde{x}_n) = S \in P. \tag{3.2}$$

The least fixed point of $\mathcal{P}$ is guaranteed to exist from the Knaster-Tarski Fixpoint theorem [32] since $\mathcal{P}$ is monotonic over $\mathbb{C}$.

*Example 2.* Let $P$ be a program composed by the declaration $f(\tilde{x}, \tilde{y}) = \textit{if } \tilde{x} > 1 \textit{ then } 3 \textit{ elseif } \tilde{y} \leq 2 \textit{ then } \tilde{x} \tilde{+} \tilde{y} \textit{ else } \tilde{x} \tilde{/} \tilde{y}$. The semantics of $P$ is defined as $\mathcal{F}[\![P]\!] = \bigcup_{i=1}^{3} \{s_i\} \cup \bigcup_{i=1}^{6} \{u_i\}$ where the conditional error bounds $s_i$ corresponding to the stable cases are:

$$s_1 = \langle R_{\mathbb{B}}(\tilde{x} > 1), \tilde{x} > 1 \rangle_{\mathbf{s}} \twoheadrightarrow (R(3), 0)^1$$

$$s_2 = \langle R_{\mathbb{B}}(\neg(\tilde{x} > 1)) \wedge R_{\mathbb{B}}(\tilde{y} \leq 2), \neg(\tilde{x} > 1) \wedge \tilde{y} \leq 2\rangle_{\mathbf{s}} \twoheadrightarrow (\chi_r(\tilde{x}) + \chi_r(\tilde{y}), \epsilon_{\dot{+}}(\chi_r(\tilde{x}), \chi_e(\tilde{x}), \chi_r(\tilde{y}), \chi_e(\tilde{y})))^{01}$$

$$s_3 = \langle R_{\mathbb{B}}(\neg(\tilde{x} > 1)) \wedge R_{\mathbb{B}}(\neg(\tilde{y} \leq 2)) \wedge \chi_r(\tilde{x} \neq 0), \neg(\tilde{x} > 1) \wedge \neg(\tilde{y} \leq 2) \wedge \tilde{x} \neq 0\rangle_{\mathbf{s}} \twoheadrightarrow (\chi_r(\tilde{x}) / \chi_r(\tilde{y}), \epsilon_{\dot{/}}(\chi_r(\tilde{x}), \chi_e(\tilde{x}), \chi_r(\tilde{y}), \chi_e(\tilde{y})))^{00}$$

The conditional error bounds modeling unstable cases $u_i$ are six and represent all the cases when real and floating-point flows diverge. For instance: $u_1 = \langle R_{\mathbb{B}}(\tilde{x} > 1), \neg(\tilde{x} > 1) \wedge \tilde{y} \leq 2\rangle_{\mathbf{u}} \twoheadrightarrow (R(3), |R(3) - (\chi_r(\tilde{x}) + \chi_r(\tilde{y}))| + \epsilon_{\dot{+}}(\chi_r(\tilde{x}), \chi_e(\tilde{x}), \chi_r(\tilde{y}), \chi_e(\tilde{y})))^{\varepsilon}$ models a case in which the outermost conditional is unstable, and $u_2 = \langle R_{\mathbb{B}}(\neg(\tilde{x} > 1)) \wedge R_{\mathbb{B}}(\tilde{y} \leq 2), \neg(\tilde{x} > 1) \wedge \neg(\tilde{y} \leq 2)\rangle_{\mathbf{u}} \twoheadrightarrow (\chi_r(\tilde{x}) + \chi_r(\tilde{y}), |\chi_r(\tilde{x}) + \chi_r(\tilde{y}) - (\chi_r(\tilde{x}) / \chi_r(\tilde{y}))| + \epsilon_{\dot{/}}(\chi_r(\tilde{x}), \chi_e(\tilde{x}), \chi_r(\tilde{y}), \chi_e(\tilde{y})))^{\varepsilon}$ models a similar case for the inner conditional.

## 4 Abstraction Scheme

The semantics presented in Section 3 is not computable since the least fixed point of the operator defined in Equation (3.2) does not converge in a finite number of steps for recursive programs. In addition, the sound treatment of unstable tests provokes an explosion of the number of semantic elements generated when several nested if-then-else occur in a function. To overcome these problems, this section presents an abstraction framework for the semantics of Section 3 that limits the combinatory explosion due to nested if-then-else expressions. A widening operator is also defined to ensure the convergence of the analysis of recursive programs. This abstraction framework yields a computable abstract semantics that is suitable for the definition of a parametric static analysis of floating-point round-off errors. The proposed abstract semantics is parametric with respect to two Galois insertions:

- $(\mathbb{E}, \leq) \underset{\alpha_{\mathbb{E}}}{\overset{\gamma_{\mathbb{E}}}{\leftrightarrows}} (\dot{\mathbb{E}}, \dot{\leq})$ between (concrete) error expressions and abstract error expression in the complete lattice $(\dot{\mathbb{E}}, \dot{\leq}, \dot{\oplus}, \dot{\otimes}, \top_{\dot{\mathbb{E}}}, \bot_{\dot{\mathbb{E}}})$, where $\dot{\leq}$ is the order relation, $\dot{\oplus}$ is the least upper bound (*lub*), $\dot{\otimes}$ is the greatest lower bound (*glb*), $\top_{\dot{\mathbb{E}}}$ is the top, and $\bot_{\dot{\mathbb{E}}}$ is the bottom of the domain.
- $(\wp(\mathbb{B} \times \widetilde{\mathbb{B}}), \hat{\Rightarrow}) \underset{\alpha_{\mathbb{B}}}{\overset{\gamma_{\mathbb{B}}}{\leftrightarrows}} (\dot{\mathbb{B}}, \dot{\Rightarrow})$ between (concrete) conditions and abstract condition in the complete lattice $(\dot{\mathbb{B}}, \dot{\Rightarrow}, \dot{\vee}, \dot{\wedge}, \top_{\dot{\mathbb{B}}}, \bot_{\dot{\mathbb{B}}})$, where $\dot{\Rightarrow}$ is the order relation, $\dot{\vee}$ is the *lub*, $\dot{\wedge}$ is the *glb*, $\top_{\dot{\mathbb{B}}}$ is the top, and $\bot_{\dot{\mathbb{B}}}$ is the bottom.

These Galois insertions have to satisfy the following properties: $\alpha_{\mathbb{E}}(0) = \bot_{\dot{\mathbb{E}}}$, $\alpha_{\mathbb{B}}(\{false, false\}) = \bot_{\dot{\mathbb{B}}}$, and $\alpha_{\mathbb{B}}(\eta_1 \hat{\wedge} \eta_2) = \alpha_{\mathbb{B}}(\eta_1) \dot{\wedge} \alpha_{\mathbb{B}}(\eta_2)$.

The abstract semantics collects approximated information and stores it in an *abstract conditional error bound*.

**Definition 6 (Abstract Conditional Error Bound).** *An* abstract conditional error bound *is defined as a tuple of the form* $\langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi}$, *where* $\dot{\eta} \in \dot{\mathbb{B}}$, $R \in \wp(\mathbb{A})$, $\dot{e} \in \dot{\mathbb{E}}$, $\pi \in Path$, *and* $t \in \{\mathbf{s}, \mathbf{u}\}$. *An abstract conditional error bound is valid when* $\dot{\eta} \not\dot{\Rightarrow} \bot_{\dot{\mathbb{B}}}$.

Abstract conditional error bounds are ordered in the following way: $\langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1} \leq \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} \iff \dot{\eta}_1 \dot{\Rightarrow} \dot{\eta}_2,\ R_1 \subseteq R_2,\ \dot{e}_1 \dot{\leq} \dot{e}_2,\ t_1 = t_2,$ and $\pi_2 \leq_{prefix} \pi_1$.

The merge (collapse) of two abstract error bounds is defined as follows.

**Definition 7.** *Let $\langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1}$ and $\langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2}$ be two abstract conditional error bounds. Their merge is defined as $\langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1} \odot \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} := \langle \dot{\eta}_1 \dot{\vee} \dot{\eta}_2 \rangle_{t_1} \twoheadrightarrow (R_1 \cup R_2, \dot{e}_1 \dot{\oplus} \dot{e}_2)^{mcp(\pi_1, \pi_2)}$ if $t_1 = t_2$, otherwise it is undefined.*

In Definition 7, the expression $mcp(\dot{\Pi})$ denotes the maximum common prefix of a set of decision paths $\dot{\Pi}$. For example, $mcp(\{0 \cdot 1 \cdot 0 \cdot 1, 0 \cdot 1 \cdot 0 \cdot 0, 0 \cdot 1\}) = 0 \cdot 1$.

As already mentioned, the concrete semantics of Section 3 computes one conditional error bound for every possible combination of real and floating-point execution path. This guarantees a sound treatment of unstable tests, but four different conditional error bounds are produced for each if-then-else. As a consequence, computing the semantics can become costly for programs with nested if-then-else expressions since the number of computed semantics elements grows exponentially. To overcome this limitation, an abstraction function is introduced to approximate sets of (concrete) conditional error bounds into sets of abstract ones. The main idea behind this abstraction is that the semantics is precisely computed just for a finite set of decision paths of interests, which are given as an input of the analysis. The conditional error bounds that correspond to other decision paths are collapsed together. Since, in general, the errors associated to unstable cases are several order of magnitude bigger than the ones due to floating-point rounding, stable and unstable cases are collapsed separately. This way, the abstraction does not lose too much precision.

The semantics presented in Section 3 is able to compute the conditions under which an unstable test occurs and to bound the error due to the difference between what is actually computed in the floating-point execution and what should have been computed in the ideal execution on real numbers. In general, this difference is large and, most of the times, one is interested just in knowing if unstable tests can occur in a program and under which circumstances. For this reason, the proposed abstraction collapses the unstable conditional error bounds in a unique expression. Using this approach, the abstract semantics is still able to soundly deal with unstable tests and to provide a sound approximation of the conditions under which the instability occurs. It also avoids the burden of differentiating each possible combination of real and floating-point paths that leads to an unstable test.

Given $\dot{\Pi} \in \wp(Path)$, let $\dot{\mathbb{C}}_{\dot{\Pi}}$ be the domain composed of sets of abstract conditional error bounds $\dot{C}$ such that for all $\langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi} \in \dot{C}$ the following properties hold.

1. If there exists $\pi' \in \dot{\Pi}$ such that $\pi' \leq_{prefix} \pi$ then the cardinality of $R$ is 1.
2. If $t = \mathbf{s}$ and there is no element in $\dot{C}$ of the form $\langle \dot{\eta}' \rangle_{\mathbf{s}} \twoheadrightarrow (R', \dot{e}')^{\pi'}$ different from $\langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi}$ such that for all $\pi'' \in \dot{\Pi}$, it holds that $\pi'' \not\leq_{prefix} \pi'$.

3. If $t = \mathbf{u}$, then there is no another unstable element in $\dot{C}$ of the form $\langle \dot{\eta}' \rangle_{\mathbf{u}} \twoheadrightarrow (R', \dot{e}')^{\pi'}$ different from $\langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi}$.

Sets of abstract conditional error bounds in $\dot{\mathbb{C}}_{\dot{\Pi}}$ are (partially) ordered as follows. For all $\dot{C}_1, \dot{C}_2 \in \dot{\mathbb{C}}_{\dot{\Pi}}$, $\dot{C}_1 \dot{\sqsubseteq} \dot{C}_2$ if and only if for all $\dot{c}_1 \in \dot{C}_1$ $\exists \dot{c}_2 \in \dot{C}_2$. $\dot{c}_1 \trianglelefteq \dot{c}_2$. The equivalence relation derived from $\dot{\sqsubseteq}$ is defined as $\dot{C}_1 \dot{\equiv} \dot{C}_2$ if and only if $\dot{C}_1 \dot{\sqsubseteq} \dot{C}_2 \wedge \dot{C}_2 \dot{\sqsubseteq} \dot{C}_1$. In the following, by abuse of notation, the quotient of $\dot{\sqsubseteq}$ over equivalence classes will be denoted with the same symbol. Furthermore, sets of conditional error bounds will be used modulo $\dot{\equiv}$ and their class will be denoted as $\dot{\mathbb{C}}_{\dot{\Pi}}$. Given $\dot{C}_1, \dot{C}_2 \in \dot{\mathbb{C}}_{\dot{\Pi}}$, their least upper bound is defined as follows

$$\begin{aligned} \dot{C}_1 \dot{\sqcup} \dot{C}_2 \coloneqq\ & [\bigcup \{ \langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi} \in \dot{C}_1 \cup \dot{C}_2 \mid \exists \pi' \in \dot{\Pi}.\ \pi' \leq_{prefix} \pi,\ t = \mathbf{s} \}]_{\dot{\equiv}} \cup \\ & \dot{\bigodot} \{ \langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi} \in \dot{C}_1 \cup \dot{C}_2 \mid \nexists \pi' \in \dot{\Pi}.\ \pi' \leq_{prefix} \pi,\ t = \mathbf{s} \} \cup \\ & \dot{\bigodot} \{ \langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi} \in \dot{C}_1 \cup \dot{C}_2 \mid t = \mathbf{u} \} \end{aligned} \tag{4.1}$$

The tuple $(\dot{\mathbb{C}}_{\dot{\Pi}}, \dot{\sqsubseteq}, \dot{\sqcup}, \dot{\sqcap}, \top_{\dot{\mathbb{C}}_{\dot{\Pi}}}, \varnothing)$ is a complete lattice, where $\top_{\dot{\mathbb{C}}_{\dot{\Pi}}} \coloneqq \bigcup \{ \langle \top_{\dot{\mathbb{B}}} \rangle_t \twoheadrightarrow (\mathbb{R}, \top_{\dot{\mathbb{E}}})^{\varepsilon} \mid t \in \{\mathbf{u}, \mathbf{s}\} \}$ is the greatest element of $\dot{\mathbb{C}}_{\dot{\Pi}}$, $\varnothing$ is the least element, and the greatest lower bound ($\dot{\sqcap}$) is defined as follows $\dot{C}_1 \dot{\sqcap} \dot{C}_2 \coloneqq [\{ \dot{c} \in \dot{\mathbf{C}} \mid \exists \dot{c}_1 \in \dot{C}_1 . \dot{c} \trianglelefteq \dot{c}_1,\ \exists \dot{c}_2 \in \dot{C}_2 . \dot{c} \trianglelefteq \dot{c}_2 \}]_{\dot{\equiv}}$.

Given $\dot{\Pi} \in \wp(Path)$, the abstraction function $\alpha_{\dot{\Pi}}$ collapses together all the stable abstract conditional error bounds that are not produced from a path in $\dot{\Pi}$. In addition, it collapses all the unstable conditional error bounds in a unique one. The abstraction function $\alpha_{\dot{\Pi}}$ and its adjoint $\gamma_{\dot{\Pi}}$ are defined as follows and form a Galois insertion $(\mathbb{C}, \sqsubseteq) \underset{\alpha_{\dot{\Pi}}}{\overset{\gamma_{\dot{\Pi}}}{\leftrightarrows}} (\dot{\mathbb{C}}_{\dot{\Pi}}, \dot{\sqsubseteq})$.

**Definition 8.** *Let $\dot{\Pi} \in \wp(Path)$, $C \in \mathbb{C}$ and $\dot{C} \in \dot{\mathbb{C}}_{\dot{\Pi}}$, the abstraction and concretization functions are defined as follows.*

$$\begin{aligned} \alpha_{\dot{\Pi}}(C) \coloneqq\ & \dot{\bigsqcup} \{ \langle \alpha_{\mathbb{B}}(\eta) \rangle_t \twoheadrightarrow (\{r\}, \alpha_{\mathbb{E}}(e))^{\pi} \mid \langle \eta \rangle_t \twoheadrightarrow (r, e)^{\pi} \in C, \\ & \qquad \exists \pi' \in \dot{\Pi} . \pi' \leq_{prefix} \pi,\ t = \mathbf{s} \} \ \dot{\sqcup} \\ & \dot{\bigodot} \{ \langle \alpha_{\mathbb{B}}(\eta) \rangle_t \twoheadrightarrow (\{r\}, \alpha_{\mathbb{E}}(e))^{\pi} \mid \langle \eta \rangle_t \twoheadrightarrow (r, e)^{\pi} \in C, \\ & \qquad \nexists \pi' \in \dot{\Pi} . \pi' \leq_{prefix} \pi,\ t = \mathbf{s} \} \ \dot{\sqcup} \\ & \dot{\bigodot} \{ \langle \alpha_{\mathbb{B}}(\eta) \rangle_t \twoheadrightarrow (\{r\}, \alpha_{\mathbb{E}}(e))^{\pi} \mid \langle \eta \rangle_t \twoheadrightarrow (r, e)^{\pi} \in C,\ t = \mathbf{u} \} \\ \gamma_{\dot{\Pi}}(\dot{C}) \coloneqq\ & \bigsqcup \{ \langle \gamma_{\mathbb{B}}(\dot{\eta}) \rangle_t \twoheadrightarrow (r, \gamma_{\mathbb{E}}(\dot{e}))^{\pi} \mid \exists \langle \dot{\eta} \rangle_t \twoheadrightarrow (R, \dot{e})^{\pi} \in \dot{C},\ r \in R \} \end{aligned}$$

**Lemma 1.** *Given $\dot{\Pi} \in \wp(Path)$, the pair of functions $(\alpha_{\dot{\Pi}}, \gamma_{\dot{\Pi}})$ is a Galois insertion between $(\mathbb{C}, \sqsubseteq)$ and $(\dot{\mathbb{C}}_{\dot{\Pi}}, \dot{\sqsubseteq})$.*

Given $\dot{\Pi} \in \wp(Path)$, an *abstract environment* is defined as a function mapping a variable to a set of abstract conditional error bounds, i.e., $\dot{Env}_{\dot{\Pi}} = \widetilde{\mathbb{V}} \to \dot{\mathbb{C}}_{\dot{\Pi}}$. The empty abstract environment is denoted as $\bot_{\dot{Env}}$ and maps every variable to the empty set $\varnothing$.

Given $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, an *abstract interpretation* is a function $\dot{I}$ such that $\forall f(\tilde{x}_i)_{i=1}^n \in \mathbb{M}$, $\dot{I}(f(\tilde{x}_i)_{i=1}^n) \in \dot{\mathbb{C}}_{\bar{\Pi}(f(\tilde{x}_i)_{i=1}^n)}$ modulo variance. The set of all interpretations respecting the aforementioned property is denoted as $\dot{\mathbb{I}}_{\bar{\Pi}}$. The empty interpretation is denoted as $\dot{\perp}_{\dot{\mathbb{I}}_{\bar{\Pi}}}$ and maps everything to the empty set. The Galois insertion of Definition 8 can be lifted to the interpretation level in the following way.

**Definition 9.** *Let $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, given $I \in \mathbb{I}$ and $\dot{I} \in \dot{\mathbb{I}}_{\bar{\Pi}}$, the abstraction function for interpretations and its adjoint are defined as follows for every function $f(\tilde{x})_{i=1}^n$ defined in $I$.*

$$\bar{\alpha}_{\bar{\Pi}}(I)(f(\tilde{x})_{i=1}^n) \coloneqq \alpha_{\bar{\Pi}(f(\tilde{x})_{i=1}^n)}(I(f(\tilde{x})_{i=1}^n))$$
$$\bar{\gamma}_{\bar{\Pi}}(\dot{I})(f(\tilde{x})_{i=1}^n) \coloneqq \gamma_{\bar{\Pi}(f(\tilde{x})_{i=1}^n)}(\dot{I}(f(\tilde{x})_{i=1}^n))$$

**Lemma 2.** *Given $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, $(\bar{\alpha}_{\bar{\Pi}}, \bar{\gamma}_{\bar{\Pi}})$ is a Galois insertion between $(\mathbb{I}, \sqsubseteq)$ and $(\dot{\mathbb{I}}_{\bar{\Pi}}, \dot{\sqsubseteq})$, where $\sqsubseteq$ and $\dot{\sqsubseteq}$ denotes the natural extension of these order relations to interpretations.*

Given $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, abstract interpretation theory [6] defines the best correct abstract version of the semantic operator $\mathcal{P}$ with respect to the Galois insertion $(\alpha_{\dot{\Pi}}, \gamma_{\dot{\Pi}})$ simply as the composition $\alpha_{\dot{\Pi}} \circ \mathcal{P} \circ \gamma_{\dot{\Pi}}$. Abstract interpretation theory [6] ensures that the abstract fixpoint semantics $\dot{\mathcal{F}}^{\dot{\Pi}} \coloneqq lfp(\dot{\mathcal{P}}^{\dot{\Pi}})$ is the best correct approximation of $\mathcal{F}$. It is correct because $\alpha_{\dot{\Pi}}(\mathcal{F}) \mathrel{\dot{\sqsubseteq}} \dot{\mathcal{F}}^{\dot{\Pi}}$ and it is the best because it is the minimum (with respect to $\dot{\sqsubseteq}$) of all correct approximations.

*Example 3.* Consider the program of Example 2 and its concrete semantics. Suppose that the selected decision path of interest is 01 and the error expressions and conditions abstraction functions are the identity. The abstract semantics of $P$ is defined as $\dot{\mathcal{F}}^{\{01\}}[\![P]\!] = s_2 \mathbin{\dot{\sqcup}} (s_1 \odot s_3) \mathbin{\dot{\sqcup}} \dot{\bigodot}_{i=1}^{6} u_i$.

The conditional error bound $s_2$, corresponding to the decision path of interest 01, is computed precisely. The other two stable bounds are collapsed together in one abstract conditional error bound of the form $s_1 \odot s_3 = \langle R_{\mathbb{B}}(\tilde{x} > 1) \vee (R_{\mathbb{B}}(\neg(\tilde{x} > 1)) \wedge R_{\mathbb{B}}(\neg(\tilde{y} \leq 2)) \wedge \chi_r(\tilde{x} \neq 0)), \tilde{x} > 1 \vee (\neg(\tilde{x} > 1) \wedge \neg(\tilde{y} \leq 2) \wedge \tilde{x} \neq 0) \rangle_{\mathbf{s}} \twoheadrightarrow (\{R(3), \chi_r(\tilde{x})/\chi_r(\tilde{y})\}, \epsilon_{\mathcal{J}}(\chi_r(\tilde{x}), \chi_e(\tilde{x}), \chi_r(\tilde{y}), \chi_e(\tilde{y})))^{\varepsilon}$. The unstable cases are collapsed together in $\dot{\bigodot}_{i=1}^{6} u_i$.

*Widening operators* [1, 6] provide a solution to the convergence problem by over-approximating infinite increasing chains in a finite number of steps. A widening operator for the domain of abstract conditional error bounds is defined. Intuitively, it approximates to the top of the domain when the recursion is possibly non terminating (the conditions are not changing and the error is growing), otherwise it tries to converge in $k$ steps for recursion calls that could terminate (the conditions are changing and they are converging to *false*).

**Definition 10.** *Given* $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, $A_1, A_2 \in \dot{\mathbb{C}}_{\dot{\Pi}}$ *such that* $\dot{C}_1 \mathrel{\dot{\sqsubseteq}} \dot{C}_2$, $n_1, n_2 \in \mathbb{N}$ *such that* $n_1 \leq n_2$, *and* $k \in \mathbb{N}$, *the operator* $\triangledown_k : (\dot{\mathbb{C}}_{\dot{\Pi}} \times \mathbb{N}) \times (\dot{\mathbb{C}}_{\dot{\Pi}} \times \mathbb{N}) \to (\dot{\mathbb{C}}_{\dot{\Pi}} \times \mathbb{N})$ *is defined as follows.*

$$
\begin{aligned}
&(\dot{C}_1, n_1) \triangledown_k (\dot{C}_2, n_2) \coloneqq \\
&\quad \Big( \dot{\bigsqcup} \{ \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \top_{\dot{\mathbb{E}}})^{\pi_2} \in \dot{C}_2 \mid \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} \in \dot{C}_2, (n_2 > k \ \textit{or} \\
&\quad (\exists \langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1} \in \dot{C}_1 \ \textit{such that} \ \dot{\eta}_1 \dot{\leftrightarrow} \dot{\eta}_2, \ R_1 \subseteq R_2 \ \textit{and} \ \dot{e}_1 \mathrel{\dot{<}} \dot{e}_2) \ \} \ \dot{\sqcup} \\
&\quad \dot{\bigsqcup} \{ \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} \in \dot{C}_2 \mid \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} \in \dot{C}_2, n_2 \leq k, \\
&\quad (\nexists \langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1} \in \dot{C}_1 \ \textit{such that} \ \dot{\eta}_1 \dot{\leftrightarrow} \dot{\eta}_2, \ R_1 \subseteq R_2 \ \textit{and} \ \dot{e}_1 \mathrel{\dot{<}} \dot{e}_2) \ \}, n_2 \Big)
\end{aligned}
$$

**Lemma 3.** *Given* $k \in \mathbb{N}$ *and* $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, *the operator* $\triangledown_k$ *is a widening operator on* $(\dot{\mathbb{C}}_{\dot{\Pi}} \times \mathbb{N})$.

Because of Lemma 3 and the results in [1,6] it is guaranteed that, for any $k \in \mathbb{N}$, $\bar{\Pi} \in [\mathbb{M} \to \wp(Path)]$, program $P \in \mathbb{P}$ and function $f(\tilde{x})_{i=1}^{n}$ defined in $P$, the chain defined as follows converges in a finite number of steps.

$$
\begin{aligned}
&(\dot{I}_0(f(\tilde{x})_{i=1}^{n}), n_0) = (\varnothing, 0) \\
&(\dot{I}_{i+1}(f(\tilde{x})_{i=1}^{n}), n_{i+1}) = \begin{cases} (\dot{I}_i(f(\tilde{x})_{i=1}^{n}), n_i) & \text{if } \dot{\mathcal{P}}^{\bar{\Pi}}[\![P]\!]_{\dot{I}_i}(f(\tilde{x})_{i=1}^{n}) \mathrel{\dot{\sqsubseteq}} I_i(f(\tilde{x})_{i=1}^{n}) \\ & \quad \text{and } n_i \leq n_{i+1} \\ (\dot{I}_i(f(\tilde{x})_{i=1}^{n}), n_i) \triangledown_k (\dot{\mathcal{P}}^{\bar{\Pi}}[\![P]\!]_{\dot{I}_i}(f(\tilde{x})_{i=1}^{n}), n_i + 1) & \\ & \quad \text{otherwise} \end{cases}
\end{aligned}
$$

## 5 PRECiSA

This section presents the prototype tool PRECiSA[4] (Program Round-off Error Certifier via Static Analysis) that implements a possible instantiation of the abstraction framework defined in Section 4. This tool is an enhancement of the tool presented in [26]. PRECiSA supports the basic arithmetic operations (addition, subtraction, multiplication, and division), square root, logarithm, exponential, trigonometric functions, floor, and absolute value. As illustrated in Fig. 1, PRECiSA accepts as inputs a program written in a simple functional language that follows the grammar in Section 4 or in PVS syntax, initial ranges for the input variables of the program, and a set of computational paths of interest for each function in the input program.

PRECiSA computes the abstract semantics presented in Section 4. The conditional error bounds corresponding to the execution paths selected by the user are computed precisely, while the others are collapsed together. A decision path of interest intuitively corresponds to a subprogram or subexpression inside a function of the input program. If the user does not select any subprogram of interest, the tool will just produce the overall round-off error for the stable case and for the unstable one.

[4] The web-interface of PRECiSA is available at `http://precisa.nianet.org`.

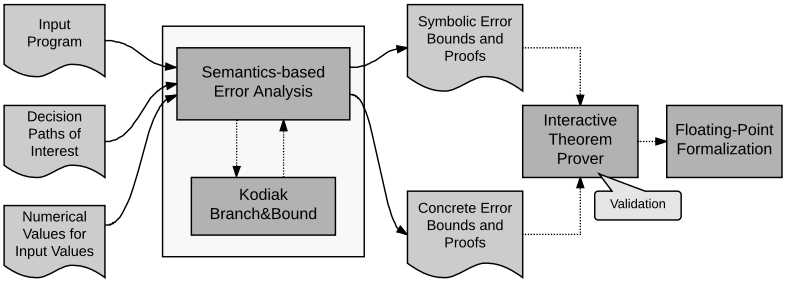


**Fig. 1.** Functional architecture of PRECiSA.

No additional abstraction is done for errors and conditions. Thus, the Galois insertions are simply defined as $(\mathbb{E}, \leq) \underset{id}{\overset{id}{\leftrightarrows}} (\mathbb{E}, \leq)$ and $(\wp(\mathbb{B} \times \widetilde{\mathbb{B}}), \hat{\Rightarrow}) \underset{id}{\overset{id}{\leftrightarrows}} (\wp(\mathbb{B} \times \widetilde{\mathbb{B}}), \hat{\Rightarrow})$. The merge of abstract conditional error bounds will be instantiated as follows $\langle \dot{\eta}_1 \rangle_{t_1} \twoheadrightarrow (R_1, \dot{e}_1)^{\pi_1} \odot \langle \dot{\eta}_2 \rangle_{t_2} \twoheadrightarrow (R_2, \dot{e}_2)^{\pi_2} := \langle \dot{\eta}_1 \hat{\vee} \dot{\eta}_2 \rangle_{t_1} \twoheadrightarrow (R_1 \cup R_2, max(\dot{e}_1, \dot{e}_2))^{mcp(\pi_1, \pi_2)}$.

The semantics presented in Section 4 is completely independent from the input values provided to the program. This makes the proposed approach scalable since it enables a compositional analysis that reuses already computed results. However, given the initial ranges for the input variables, it is essential to compute numerical bounds from the (abstract) conditional error bounds. To this aim, the proposed prototype tool uses the optimizer Kodiak [30] which is based on the formally verified branch-and-bound algorithm presented in [27]. This branch-and-bound algorithm relies on enclosure functions for arithmetic operators. These enclosure functions compute provably correct over approximations of the symbolic error expressions using either interval arithmetic or Bernstein basis. The algorithm recursively splits the domain of the function into smaller subdomains and computes an enclosure of the original expression in these subdomains. The recursion stops when a precise enclosure is found, based on a given precision, or when a given maximum recursion depth is reached. The output of the algorithm is a numerical enclosure for each symbolic error expression.

Besides computing error bounds, PRECiSA generates proof certificates ensuring that these bounds are correct. Having an externally checkable certificate increases the level of trustworthiness of the proposed tool. PRECiSA relies on the higher-order logic interactive theorem prover PVS [28] and a floating-point formalization originally presented in [3] and extended in [26]. Therefore, each computed conditional error bound is translated into a lemma stating that, provided the conditions are satisfied, the floating-point value resulting from the execution of $f$ on floating-point values differs from the exact real-number computation by at most the round-off error approximation computed by the semantics. PRECiSA generates proof scripts that automatically discharge the generated lemmas.

In the following, PRECiSA is compared in terms of accuracy and performance with the following floating-point analysis tools: Gappa (ver. 1.3.1) [11], Fluctuat

(ver. 3.1376) [15], FPTaylor (ver. 0.9) [31], Real2Float [22], and Rosa [8] (see Section 6 for a description of each tool). This comparison was performed using benchmarks taken from the Rosa and FPTaylor repositories. The selected benchmarks involve nonlinear expressions and polynomial approximations of functions, taken from equations used in physics, control theory, and biological modeling. In addition, some extra benchmarks taken from real-world avionics algorithms are considered. The experimental environment consisted of a 2.5 GHz Intel Core i7-4710MQ with 24 GB of RAM, running under Ubuntu 16.04 LTS. The benchmarks presented in this section and the corresponding proof certificates are available as part of the PRECiSA distribution.[5]

Table 1 shows numerical round-off error bounds computed by the aforementioned tools. Since the considered tools offers different configurations and options for the analysis, only the best estimation obtained by each tool for each example is reported in the table. In fact, FPTaylor offers two different optimization algorithms and two different rounding models. Gappa and Fluctuat allow the user to manually provide hints to obtain tighter error bounds. For the sake of uniformity, for all examples and tools, input variables and constants are assumed to be real numbers. This means that they carry a round-off error that has to be taken into consideration in the analysis. PRECiSA compares favorably to the other tools in terms of precision. Additionally, it supports a large set of basic and transcendental operators as well as common programming languages constructs such as conditionals and loops. On the contrary, some of the other tools lack that support, hence, they cannot analyze all the benchmarks. For instance, the floor operator appears in the cpr_yz0 and it is not supported by Real2Float, Rosa, and FPTaylor. Stynlinski and PolyCARP contain conditionals that are not handled by FPTaylor and Gappa. PRECiSA is the only tool that is able to analyze the recursive program mult_pow2_rec.

The times for the computation of the bounds in Table 1 are shown in Table 2. Overall, Fluctuat is the fastest approach but it does not produce certificates for the soundness of its results. The performance of PRECiSA is in line with similar tools for most of the examples, and for some of the considered benchmarks PRECiSA is the fastest approach.

In summary, for the considered examples, the proposed tool provides a good trade-off between accuracy and performance together with a wide support for arithmetic operations and programming constructs.

## 6 Related Work

The use of abstract interpretation and semantics based approaches for the problem of analyzing floating-point programs is not new. The static analyzer Astrée [7] automatically detects the presence of potential floating-point run-time exceptions such as overflows by means of sound floating-point abstract domains [4,25]. The abstraction scheme presented here shares some similarities with the ap-

[5] The PRECiSA distribution is available at `https://github.com/nasa/PRECiSA`.

| | Gappa | Fluctuat | Real2Float | Rosa | FPTaylor | PRECiSA |
|---|---|---|---|---|---|---|
| azimuth | n/a | n/a | 2.83E-13 | n/a | **8.32E-15** | *1.19E-13* |
| carbonGas | *6.01E-09* | 1.17E-08 | 2.21E-08 | 1.60E-08 | **5.90E-09** | 7.17E-09 |
| doppler1 | 1.61E-13 | *1.27E-13* | 7.65E-12 | 2.68E-13 | **1.22E-13** | 1.98E-13 |
| doppler2 | 2.86E-13 | *2.35E-13* | 1.57E-11 | 6.45E-13 | **2.23E-13** | 3.81E-13 |
| doppler3 | 8.69E-14 | *7.12E-14* | 8.59E-12 | 1.01E-13 | **6.63E-14** | 1.09E-13 |
| himmilbeau | **8.51E-13** | *1.00E-12* | 1.42E-12 | *1.00E-12* | *1.00E-12* | *1.00E-12* |
| jet | 4.45E+03 | 1.07E-10 | n/a | 4.91E-09 | **1.03E-11** | *1.59E-11* |
| kepler0 | 1.09E-13 | 1.03E-13 | 1.20E-13 | *8.28E-14* | **7.47E-14** | 1.06E-13 |
| kepler1 | 4.68E-13 | *3.51E-13* | 4.67E-13 | 4.14E-13 | **2.86E-13** | 3.90E-13 |
| kepler2 | 2.38E-12 | 2.24E-12 | 2.09E-12 | 2.15E-12 | *1.58E-12* | **1.53E-12** |
| predatorPrey | *1.67E-16* | 2.35E-16 | 2.51E-16 | 1.98E-16 | **1.59E-16** | 1.84E-16 |
| rigidBody1 | **2.95E-13** | *3.22E-13* | 5.33E-13 | *3.22E-13* | **2.95E-13** | **2.95E-13** |
| rigidBody2 | *3.61E-11* | 3.65E-11 | 6.48E-11 | 3.65E-11 | *3.61E-11* | **3.60E-11** |
| sine | 6.91E-16 | 7.41E-16 | 6.03E-16 | *5.18E-16* | **3.87E-16** | 6.37E-16 |
| sineOrder3 | *6.54E-16* | 1.09E-15 | 1.19E-15 | 9.96E-16 | **5.94E-16** | 1.17E-15 |
| sphere | n/a | n/a | 1.52E-14 | n/a | **8.11E-15** | *9.99E-15* |
| sqroot | 5.35E-16 | 6.83E-16 | 1.28E-15 | 6.18E-16 | *5.01E-16* | **4.29E-16** |
| t_div_t1 | 9.99E+00 | 2.80E-12 | *8.53E-16* | 5.68E-11 | **2.22E-16** | 3.91E-15 |
| turbine1 | 2.41E-14 | 3.09E-14 | 2.46E-11 | 5.99E-14 | **1.66E-14** | *2.17E-14* |
| turbine2 | 3.32E-14 | *2.59E-14* | 2.07E-12 | 7.67E-14 | **1.99E-14** | 2.81E-14 |
| turbine3 | 3.52E-01 | 1.34E-14 | 1.70E-11 | 4.62E-14 | **9.55E-15** | *1.22E-14* |
| verhulst | *2.84E-16* | 4.80E-16 | 4.66E-16 | 4.67E-16 | **2.47E-16** | 3.74E-16 |
| PolyCARP (stable) | n/a | *1.89E-15* | n/a | n/a | n/a | **1.83E-15** |
| PolyCARP (unstable) | n/a | n/a | 6.60E+00 | n/a | n/a | **6.00E-01** |
| Stynlinski (stable) | n/a | **2.29E-14** | n/a | *2.31E-14* | n/a | 4.28E-14 |
| Stynlinski (unstable) | n/a | **2.29E-14** | n/a | *2.31E-14* | n/a | 1.61E+02 |
| cpr_yz0 | *1.35E+05* | **1.31E+05** | n/a | n/a | n/a | **1.31E+05** |
| logExp | n/a | n/a | *2.52E-15* | n/a | **1.49E-15** | 3.22E-15 |
| hartman3 | n/a | n/a | 2.99E-13 | n/a | **3.26E-15** | *1.58E-14* |
| hartman6 | n/a | n/a | 5.07E-13 | n/a | **5.26E-15** | *2.24E-13* |
| mult_pow2_rec (stable) | n/a | n/a | n/a | n/a | n/a | **7.11E-15** |

**Table 1.** Experimental results for absolute round-off error bounds (**bold** indicates the best approximation, *italic* indicates the second best.)

proach of [19] where the analysis is refined by partitioning the program with respect to its control flow.

Some semantics-based approaches have been proposed to estimate the round-off error of a program. In [23], a family of abstract semantics parametric with respect to the error order and to a partition of the program is proposed for floating-point round-off errors. In [17], several abstract semantics for the static analysis of finite precision computations are defined. In contrast to the approach presented in this paper, the abstract semantics in [23] and [17] are not compositional since in these approaches the error is computed starting from a set of input ranges for the initial variables.

Diverse analysis techniques and tools to estimate the round-off error of floating-point computations have been proposed in the literature. Fluctuat [15] is a commercial analyzer that accepts as input a C program with annotations about input bound and uncertainties, and it produces bounds for the round-off error of the program expressions decomposed with respect to its provenance. Fluctuat uses a zonotopic abstract domain [17] that is based on affine arithmetic [12]. It is able to soundly treat unstable tests as explained in [18] and it provides support for iterative programs by using the widening operators introduced in [13, 16]. The widening operator presented in this paper is different from the ones of [13, 16] in that it takes advantage of the information contained in the path conditions.

| | Gappa | Fluctuat | Real2Float | Rosa | FPTaylor | PRECiSA |
|---|---|---|---|---|---|---|
| azimuth | n/a | n/a | **1.986** | n/a | 26.050 | *5.204* |
| carbonGas | 2.130 | **0.062** | 0.776 | 26.734 | 0.497 | *0.090* |
| doppler1 | 3.475 | 6.904 | 5.957 | 17.293 | *1.280* | **0.447** |
| doppler2 | 3.456 | 6.835 | 5.934 | 18.336 | *1.504* | **0.402** |
| doppler3 | 3.604 | 6.837 | 5.846 | 26.996 | *1.449* | **0.419** |
| himmilbeau | 1.636 | **0.013** | 0.193 | 4.478 | 0.473 | *0.106* |
| jet | 8.604 | **1.033** | n/a | 264.811 | *2.457* | 255.278 |
| kepler0 | 8.107 | 9.467 | **0.203** | 3.377 | *2.813* | 2.878 |
| kepler1 | 2.088 | **0.430** | 8.509 | 132.313 | *1.642* | 8.964 |
| kepler2 | 9.303 | *2.233* | 6.630 | 63.256 | **0.743** | 345.785 |
| predatorPrey | 1.259 | **0.019** | 0.684 | 26.452 | 0.521 | *0.021* |
| rigidBody1 | *0.030* | **0.013** | 0.434 | 0.298 | 0.427 | 0.049 |
| rigidBody2 | *0.047* | **0.014** | 0.272 | 2.752 | 0.470 | 1.035 |
| sine | 4.147 | **0.022** | 0.872 | 4.513 | *0.625* | 0.631 |
| sineOrder3 | 1.966 | **0.017** | 0.296 | 0.771 | 0.437 | *0.021* |
| sphere | n/a | n/a | *0.033* | n/a | 35.116 | **0.020** |
| sqroot | 4.968 | **0.014** | 0.713 | 1.328 | 43.428 | *0.047* |
| t_div_t1 | 0.160 | **0.017** | 34.656 | 6.207 | 0.418 | *0.021* |
| turbine1 | 6.222 | *5.410* | 67.599 | 19.254 | 62.760 | **1.746** |
| turbine2 | 4.185 | 4.311 | *3.927* | 6.483 | 44.138 | **2.003** |
| turbine3 | 6.927 | *5.417* | 66.991 | 20.642 | 62.623 | **4.569** |
| verhulst | 0.346 | **0.018** | 0.425 | 7.730 | 0.418 | *0.019* |
| Polycarp (stable) | n/a | **0.013** | n/a | n/a | n/a | *0.018* |
| Polycarp (unstable) | n/a | n/a | *0.024* | n/a | n/a | **0.018** |
| Stynlinski (stable) | n/a | **0.266** | n/a | 58.543 | n/a | *16.376* |
| Stynlinski (unstable) | n/a | **0.313** | n/a | 58.543 | n/a | *16.376* |
| yz0 | 7.177 | **0.014** | n/a | n/a | n/a | *0.249* |
| logExp | n/a | n/a | 0.664 | n/a | *0.389* | **0.026** |
| hartman3 | n/a | n/a | **1.760** | n/a | 84.147 | *44.309* |
| hartman6 | n/a | n/a | **87.582** | n/a | *2191.622* | 4320.212 |
| mult_pow2_rec (stable) | n/a | n/a | n/a | n/a | n/a | **0.037** |

**Table 2.** Times in seconds for the generation of round-off error bounds and certificates (**bold** indicates the best time, *italic* indicates the second best.)

RangeLab [24] is an interactive tool that determines the range of the round-off errors for elementary arithmetic expression based on the semantics of [23]. RangeLab is able to deal with while loops by means of a widening operator based on the classical interval domain widening. However, it does not provides a sound approximation of unstable conditionals. RangeLab and Fluctuat do not generate formal certificates for the computed bounds and they are not compositional.

FPTaylor [31] uses symbolic Taylor expansions to approximate floating-point expressions and applies a global optimization technique to obtain tight bounds for round-off errors. In addition, FPTaylor emits certificates for HOL Light [20] except for the configurations that use an improved rounding model that correlates error terms and allows much tighter error bounds [2]. Because of the technique used by FPTaylor, it is restricted to smooth functions. Unlike PRECiSA, which targets programs with conditional and function calls, FPTaylor is designed to analyze arithmetic expressions. FPTuner [5] uses FPTaylor to implement a rigorous approach to precision allocation of mixed-precision arithmetic expressions.

VCFloat [29] is a tool that automatically computes round-off error terms for numerical C expressions along with their correctness proof in Coq. This tool uses interval arithmetic to approximate the error bounds and generates validity conditions on the expressions. Similarly to FPTaylor, VCFloat targets only

arithmetic expressions. Real2Float [22] computes certified bounds for round-off errors by using an optimization technique employing semidefinite programming and sum of square certificates. Real2Float at the moment does not handle denormal floating-point numbers nor loops. Gappa [11] computes enclosures for floating-point expressions via interval arithmetic. This enclosure method enables a quick computation of the bounds, but may result in pessimistic error estimations. Gappa also generates a proof of the results that can be checked in the Coq proof assistant. In Gappa, the bound computation, the certification construction, and their verification may require hints from the user. Thus, some level of expertise is required, unlike PRECiSA, which is fully automatic. Rosa [8, 9] uses a compilation algorithm that, from an ideal real-valued implementation, produces a finite-precision version (if it exists) that is guaranteed to meet a given desired precision. Rosa soundly deals with unstable tests and with bounded loops with bounded variables.

## 7 Conclusion

In this paper, a semantic framework based on abstract interpretation has been presented with the aim of providing a parametric round-off error static analysis for floating-point programs. The abstract semantics defined by this framework enjoys several features. It is defined in a compositional way, which allows for an incremental, modular, and efficient treatment of the program being analyzed. This makes the analysis defined upon this framework scalable and reusable. Moreover, the semantics is able to deal with any floating-point operator provided the existence of a round-off error estimation that satisfies Formula (2.5). Finally, recursion and conditionals are soundly handled.

The semantic analysis proposed in this paper is sound with respect to unstable tests and it associates conditions to the computed error estimation. This makes the analysis more precise since different execution paths may lead to different round-off errors. The proposed technique also avoids considering computations that lead to runtime errors such as division by zero or square root of a negative number. Additionally, the information collected in the conditions is used to discard impossible execution paths and to characterize initial input values that may cause large round-off errors.

PRECiSA is an implementation of the proposed framework that, additionally, generates proof certificates ensuring the correctness of the computed error bounds. In future work, the authors plan to integrate in PRECiSA other abstract domains such as affine arithmetic and a compositional version of the symbolic Taylor expansions of [31]. This way, the most suitable domain can be chosen depending on the input program and on the desired tradeoff between efficiency and precision. Another interesting future direction is the integration of PRECiSA with the static analyzer Frama-C [21]. This integration will enable the automated formal verification of C floating-point programs.

## References


1. Bagnara, R., Hill, P.M., Ricci, E., Zaffanella, E.: Precise Widening Operators for Convex Polyhedra. Science of Computer Programming 58(1-2), 28–56 (2005)
2. Baranowski, M., Briggs, I., Chiang, W., Gopalakrishnan, G., Rakamaric, Z., Solovyev, A.: Moving the Needle on Rigorous Floting-point Precision Tuning. 6th Workshop on Automated Formal Methods (AFM 2017) (2017)
3. Boldo, S., Muñoz, C.: A high-level formalization of floating-point numbers in PVS. Tech. Rep. CR-2006-214298, NASA (2006)
4. Chen, L., Miné, A., Cousot, P.: A sound floating-point polyhedra abstract domain. In: Proceedings of the 6th Asian Symposium on Programming Languages and Systems, APLAS 2008. Lecture Notes in Computer Science, vol. 5356, pp. 3–18. Springer (2008)
5. Chiang, W., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., Rakamarić, Z.: Rigorous floating-point mixed-precision tuning. In: Proceedings of POPL 2017. pp. 300–315. ACM (2017)
6. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proceedings of POPL 1977. pp. 238–252. ACM (1977), http://doi.acm.org/10.1145/512950.512973
7. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival: The ASTREÉ Analyzer. In: Proceedings of the 14th European Symposium on Programming (ESOP 2005). Lecture Notes in Computer Science, vol. 3444, pp. 21–30. Springer (2005)
8. Darulova, E., Kuncak, V.: Sound compilation of reals. In: Proceedings of POPL 2014. pp. 235–248. ACM (2014), http://doi.acm.org/10.1145/2535838.2535874
9. Darulova, E., Kuncak, V.: Towards a compiler for reals. ACM Transactions on Programming Languages and Systems 39(2), 8:1–8:28 (2017)
10. Daumas, M., Rideau, L., Théry, L.: A generic library for floating-point numbers and its application to exact computing. In: Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics". pp. 169–184. Springer Berlin Heidelberg (2001)
11. de Dinechin, F., Lauter, C., Melquiond, G.: Certifying the floating-point implementation of an elementary function using Gappa. IEEE Trans. on Computers 60(2), 242–253 (2011), http://dx.doi.org/10.1109/TC.2010.128
12. de Figueiredo, L.H., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms 37(1-4), 147–158 (2004), http://dx.doi.org/10.1023/B:NUMA.0000049462.70970.b6
13. Ghorbal, K., Goubault, E., Putot, S.: A logical product approach to zonotope intersection. In: Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010. Lecture Notes in Computer Science, vol. 6174, pp. 212–226. Springer (2010), http://dx.doi.org/10.1007/978-3-642-14295-6_22
14. Goldberg, D.: What Every Computer Scientist Should Know About Floating-point Arithmetic. ACM Comput. Surv. 23(1), 5–48 (1991)
15. Goubault, E., Putot, S.: Static analysis of numerical algorithms. In: Proceedings of SAS 2006. Lecture Notes in Computer Science, vol. 4134, pp. 18–34. Springer (2006), http://dx.doi.org/10.1007/11823230_3
16. Goubault, E., Putot, S.: Perturbed affine arithmetic for invariant computation in numerical program analysis. CoRR abs/0807.2961 (2008)

17. Goubault, E., Putot, S.: Static analysis of finite precision computations. In: Proceedings of VMCAI 2011. Lecture Notes in Computer Science, vol. 6538, pp. 232–247. Springer (2011), http://dx.doi.org/10.1007/978-3-642-18275-4_17
18. Goubault, E., Putot, S.: Robustness analysis of finite precision implementations. In: Proceedings of APLAS 2013. Lecture Notes in Computer Science, vol. 8301, pp. 50–57. Springer (2013), http://dx.doi.org/10.1007/978-3-319-03542-0_4
19. Handjieva, M., Tzolovski, S.: Refining static analyses by trace-based partitioning using control flow. In: Proceedings of the 5th International Symposium on Static Analysis (SAS '98). pp. 200–214 (1998)
20. Harrison, J.: HOL light: An overview. In: Proceedings of TPHOLs 2009. Lecture Notes in Computer Science, vol. 5674, pp. 60–66. Springer (2009), http://dx.doi.org/10.1007/978-3-642-03359-9_4
21. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c: A software analysis perspective. Formal Aspects of Computing 27(3), 573–609 (2015)
22. Magron, V., Constantinides, G., Donaldson, A.: Certified roundoff error bounds using semidefinite programming. ACM Trans. Math. Softw. 43(4), 34:1–34:31 (2017)
23. Martel, M.: Semantics of roundoff error propagation in finite precision calculations. Higher-Order and Symbolic Computation 19(1), 7–30 (2006), http://dx.doi.org/10.1007/s10990-006-8608-2
24. Martel, M.: RangeLab: A static-analyzer to bound the accuracy of finite-precision computations. In: Proceedings of SYNASC 2011. pp. 118–122. IEEE Computer Society (2011), http://dx.doi.org/10.1109/SYNASC.2011.52
25. Miné, A.: Relational abstract domains for the detection of floating-point run-time errors. In: Proceedings of the 13th European Symposium on Programming Languages and Systems, ESOP 2004. Lecture Notes in Computer Science, vol. 2986, pp. 3–17. Springer (2004), http://dx.doi.org/10.1007/978-3-540-24725-8_2
26. Moscato, M.M., Titolo, L., Dutle, A., Muñoz, C.: Automatic estimation of verified floating-point round-off errors via static analysis. In: Proceedings of the International Conference on Computer Safety, Reliablilty, and Security, SAFECOMP 2017 (2017)
27. Narkawicz, A., Muñoz, C.: A formally verified generic branching algorithm for global optimization. In: Revised Selected Papers of VSTTE 2013. Lecture Notes in Computer Science, vol. 8164, pp. 326–343. Springer (2013), http://dx.doi.org/10.1007/978-3-642-54108-7_17
28. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Proceedings of CADE 1992. Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer (1992)
29. Ramananandro, T., Mountcastle, P., Meister, B., Lethin, R.: A unified coq framework for verifying C programs with floating-point computations. In: Proceedings of CPP 2016. pp. 15–26. ACM (2016), http://doi.acm.org/10.1145/2854065.2854066
30. Smith, A.P., Muñoz, C.A., Narkawicz, A.J., Markevicius, M.: A rigorous generic branch and bound solver for nonlinear problems. In: 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2015, Timisoara, Romania, September 21-24, 2015. pp. 71–78 (2015)
31. Solovyev, A., Jacobsen, C., Rakamaric, Z., Gopalakrishnan, G.: Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions. In: Proceedings of FM 2015. Lecture Notes in Computer Science, vol. 9109, pp. 532–550. Springer (2015), http://dx.doi.org/10.1007/978-3-319-19249-9_33
32. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. In: Pacific Journal of Mathematics. pp. 285–309 (1955)