Semantics-Based Transformation of Arithmetic Expressions

Matthieu Martel

CEA - LIST
CEA F91191 Gif-Sur-Yvette Cedex, France
matthieu.martel@cea.fr

Abstract. Floating-point arithmetic is an important source of errors in programs because of the loss of precision arising during a computation. Unfortunately, this arithmetic is not intuitive (e.g. many elementary operations are not associative, inversible, etc.) making the debugging phase very difficult and empiric.

This article introduces a new kind of program transformation in order to automatically improve the accuracy of floating-point computations. We use P. Cousot and R. Cousot's framework for semantics program transformation and we propose an offline transformation. This technique was implemented, and the first experimental results are presented.

1 Introduction

In this article, we introduce a new kind of program transformation in order to improve the precision of the evaluation of expressions in floating-point arithmetic. We consider that an expression implements a formula obeying the usual laws of mathematics. This means that, in particular, the evaluation of the formula in infinite precision yields an exact result and that algebraic rules like associativity, commutativity or distributivity do not modify the meaning of a formula. However, floating-point arithmetic differs strongly from real number arithmetic: the values have a finite number of digits and the algebraic laws mentioned earlier no longer hold. Consequently, the evaluation by a computer of mathematically equivalent formulas (for example $x \times (1 + x)$ and $x + x^2$) possibly leads to very different results.

Our work is motivated by the fact that, in programs, errors due to floating-point arithmetic are very difficult to understand and to rectify. Recently, validation techniques based on abstract interpretation have been developed to assert the numerical accuracy of these calculations [13,8] but, while these tools enable one to detect the imprecisions and, possibly, to understand their origin, they do not help the programmer to correct the programs. Unfortunately, floating-point arithmetic is not intuitive, making the debugging phase very difficult and empiric: there exists no methodology to improve the accuracy of a computation and we have at most a set of tricks like "sort numbers increasingly before adding

H. Riis Nielson and G. Filé (Eds.): SAS 2007, LNCS 4634, pp. 298–314, 2007.

them" or "use Horner's method to evaluate a polynomial." Performing these transformations by hand is tedious because the computer arithmetic is subtle. Therefore, their automatization is of great practical interest. Even if static analysis techniques have already given rise to industrially usable tools to assert the numerical precision of critical codes [8,9], there is an important gap between validation and automatic correction. To our knowledge, this article is the first attempt in that new direction.

We introduce a new kind of program transformation, in order to automatically improve the "quality" of an arithmetic expression with respect to some evaluation criterion: the precision of floating-point computations. We use P. Cousot and R. Cousot's framework for semantics program transformation [6] by abstract interpretation [5] and we propose an offline transformation. The methodology of [6] enables us to define a semantics transformation that would be far more difficult to obtain at the syntactic level, since there is no strong syntactic relation between the source and transformed expressions.

For the sake of simplicity, we restrict ourselves to arithmetic expressions, neglecting, in this first work, the statements of a full programming language. However, our techniques are not specific to expressions and can be extended to complete programming languages.

The main steps of our method are the following. First, we introduce a nondeterministic small-step operational semantics for the evaluation of real expressions. Basically, algebraic laws like associativity, commutativity or distributivity make it possible to evaluate the same expression in many different ways (all confluent to the same final result.) Next, the same semantics is applied to floating-point arithmetic. In this case, different evaluations of an expression yield different results because the algebraic laws of the reals do not work any longer. Then we compute the quality of each execution path of the floating-point arithmetic based semantics by means of a non-standard domain (e.g. the global error arithmetic developed for validation of floating-point computation [13,12]). However, because there are too many paths in the previous semantics, we define a new abstract semantics in which sets of traces are merged into abstract traces. Basically, we merge traces in which sub-expressions have been evaluated approximatively in the same way, using abstract expressions of limited height. The semantics transformation then consists of computing (approximatively) the execution path which optimizes the quality of the evaluation. The correctness of the transformation stems from the fact that, at the observational level (i.e. in the reals), all the execution paths that we consider lead to the same final result. Other classical abstractions of sets of numbers by intervals is used, in order to deal with sets of values and to find the best expression for a range of inputs. A prototype has been implemented and we also present some experimental results.

The rest of this article is organized as follows: Section 2 gives an overview of our transformation and of the semantics we use. Section 3 and Section 4 introduce the concrete and abstract semantics. The transformation is presented in Section 5 and experimental results are given in Section 6. Sections 7 and 8 are dedicated to perspectives and concluding remarks.

2 Overview

As mentioned in the introduction, we aim at transforming mathematic expressions in order to improve the precision of their evaluation in floating-point arithmetic. For example, let us consider the simple formula which computes the area of a rectangular parallelepiped of dimension $a \times b \times c$:

$$A = 2 \times \big((a \times b) + (b \times c) + (c \times a)\big) \tag{1}$$

Let us consider a thin parallelepiped of dimensions $a = 1 \quad b = c = \frac{1}{9}$. With these values, the examination of Equation (1) reveals that $ab \gg ac$ and $ab \gg bc$. It is well-known that in floating-point arithmetic, adding numbers of different magnitude may lead to important precision loss: if $x \ll y$ then, possibly, $x +_{\mathbb{F}} y = y$ (this is called an absorption). In our example, absorptions arise in the direct evaluation of A. The transformation introduced in this article enables one to automatically rewrite the original expression (where $d = 2$):

```
d*(((a*b)+(b*c))+(c*a))
```

into the new expression:

```
(((c*a)*d)+(d*(b*c)))+(d*(a*b))
```

In this new formula, the smallest terms are summed first. Furthermore, the product is distributed and this avoids a multiplication of roundoff errors of the additions by a large value. This transformation relies on several semantics which are summarized below.

- $\rightarrow_{\mathbb{F}}$ is the concrete semantics based on the floating-point arithmetic. This semantics corresponds to the evaluation of an expression e by a computer. $\rightarrow_{\mathbb{F}}$ is defined in Section 3.1.
- $\rightarrow_{\mathbb{R}}$ is the concrete semantics based on real number arithmetic. In $\mathbb{R}$, algebraic rules hold, like associativity, distributivity, etc. $\rightarrow_{\mathbb{R}}$ is defined in Section 3.2.
- $\rightarrow_{\mathbb{E}}$ is the non-standard semantics based on the arithmetic of floating-point numbers with global errors. This semantics calculates the exact global error between a real computation and a floating-point computation [12].$\rightarrow_{\mathbb{E}}$ is defined in Section 3.3.
- $\longrightarrow$ is the non-standard semantics used to define our abstract interpretation. $\longrightarrow$ is defined in Section 4.1.
- $\xrightarrow{A}_k$ is the abstract semantics. k is a parameter which defines the precision of the semantics. $\xrightarrow{A}_k$ is defined in Section 4.2.

3 Concrete Semantics of Expressions

In this section, we introduce some concrete semantics of expressions, for floating-point arithmetic, for real arithmetic and for floating-point numbers with global errors (the semantics $\rightarrow_{\mathbb{F}}$, $\rightarrow_{\mathbb{R}}$ and $\rightarrow_{\mathbb{E}}$ mentioned in Section 2). $\rightarrow_{\mathbb{F}}$ is the semantics used by a computer which complies with the IEEE754 Standard [1], $\rightarrow_{\mathbb{R}}$

$$\frac{v = v_1 +_{\mathbb{F}} v_2}{v_1 + v_2 \ \rightarrow_{\mathbb{F}} \ v} \qquad \frac{e_1 \ \rightarrow_{\mathbb{F}} \ e_1'}{e_1 + e_2 \ \rightarrow_{\mathbb{F}} \ e_1' + e_2} \qquad \frac{e_2 \ \rightarrow_{\mathbb{F}} \ e_2'}{v_1 + e_2 \ \rightarrow_{\mathbb{F}} \ v_1 + e_2'}$$

$$\frac{v = v_1 \times_{\mathbb{F}} v_2}{v_1 \times v_2 \ \rightarrow_{\mathbb{F}} \ v} \qquad \frac{e_1 \ \rightarrow_{\mathbb{F}} \ e_1'}{e_1 \times e_2 \ \rightarrow_{\mathbb{F}} \ e_1' \times e_2} \qquad \frac{e_2 \ \rightarrow_{\mathbb{F}} \ e_2'}{v_1 \times e_2 \ \rightarrow_{\mathbb{F}} \ v_1 \times e_2'}$$

Fig. 1. The reduction rules for floating point arithmetic

is used in the correctness proofs where it plays the role of observer [6], and $\rightarrow_{\mathbb{E}}$ is used to define non-standard and abstract semantics of programs.

For the sake of simplicity, we only consider elementary arithmetic expressions generated by the grammar:

$$e \ ::= \ v \mid x \mid e_1 + e_2 \mid e_1 \times e_2. \tag{2}$$

In Equation (2), v denotes a value and $x \in \mathrm{Id}$ is a constant whose value is given by a global environment. These global variables are implemented in our prototype, and they introduce no theoretical difficulty. We omit them in all the formal semantics.

3.1 Floating-Point Arithmetic Based Semantics

The semantics $\rightarrow_{\mathbb{F}}$ just defines how an expression is evaluated by a computer, following the IEEE754 Standard for floating-point arithmetic.

Let $\uparrow_\circ : \mathbb{R} \rightarrow \mathbb{F}$ be the function which returns the roundoff of a real number following the rounding mode $\circ \in \{\circ_{-\infty}, \circ_{+\infty}, \circ_0, \circ_\sim\}$ [1]. $\uparrow_\circ$ is fully specified by the IEE754 Standard which also requires, for any elementary operation $\Diamond$, that:

$$x_1 \ \Diamond_{\mathbb{F},\circ} \ x_2 \ = \uparrow_\circ (x_1 \ \Diamond_{\mathbb{R}} \ x_2) \tag{3}$$

Equation (3) states that the result of an operation between floating-point numbers is the roundoff of the exact result of this operation. In this article, we also use the function $\downarrow_\circ$: $\mathbb{R} \rightarrow \mathbb{R}$ which returns the roundoff error. We have $\downarrow_\circ (r) = r - \uparrow_\circ (r)$.

The floating-point arithmetic based semantics of expressions is defined by the rules of Figure 1. This semantics is obvious but we will need it to prove the correctness of the transformation in Section 4.3.

3.2 Real Arithmetic Based Semantics

The exact evaluation of the expressions in Equation (2) is given by the real arithmetic. So, we define the reduction rules $\rightarrow_{\mathbb{R}}$ by assuming that any value v belongs to $\mathbb{R}$ and by using the reduction rules of Figure 2 in which $\oplus$ and $\otimes$ stand for $+_{\mathbb{R}}$ and $\times_{\mathbb{R}}$ (the addition and product between real numbers).

The rules of equations (4) to (7) are straightforward. The rule of Equation (8) relies on the syntactic relation $\equiv$ defined as being the smallest equivalence relation containing relations (i) to (vii) of Figure 2. The equivalence $\equiv$ identifies

$$\frac{v = v_1 \oplus v_2}{v_1 + v_2 \ \rightarrow \ v} \tag{4}$$

$$\frac{v = v_1 \otimes v_2}{v_1 \times v_2 \ \rightarrow \ v} \tag{5}$$

$$\frac{e_1 \ \rightarrow \ e_1'}{e_1 + e_2 \ \rightarrow \ e_1' + e_2} \tag{6}$$

$$\frac{e_1 \ \rightarrow \ e_1'}{e_1 \times e_2 \ \rightarrow \ e_1' \times e_2} \tag{7}$$

$$\frac{e \equiv e_1 \quad e_1 \ \rightarrow \ e_1' \quad e_1' \equiv e'}{e \ \rightarrow \ e'} \tag{8}$$

(i) $(e_1 + e_2) + e_3 \equiv e_1 + (e_2 + e_3)$
(ii) $e_1 + e_2 \equiv e_2 + e_1$
(iii) $e \equiv e + 0$
(iv) $(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3)$
(v) $e_1 \times e_2 \equiv e_2 \times e_1$
(vi) $e \equiv e \times 1$
(vii) $e_1 \times (e_2 + e_3) \equiv e_1 \times e_2 + e_1 \times e_3$

Fig. 2. The reduction rules for arithmetic expressions

arithmetic expressions which are equal in the reals, using associativity, distributivity and the neutral elements of $\mathbb{R}$. Equation (8) makes our transition system non-deterministic: there exist many reduction paths to evaluate the same expression. However, in $\mathbb{R}$, this transition system is (weakly) confluent and all the evaluations yield the same final result. This is summed up by the following property in which $\rightarrow_{\mathbb{R}}^{*}$ denotes the transitive closure of $\rightarrow_{\mathbb{R}}$.

Property 1. *Let e be an arithmetic expression. If $e \rightarrow_{\mathbb{R}} e_1$ and $e \rightarrow_{\mathbb{R}} e_2$ then there exists e' such that $e_1 \rightarrow_{\mathbb{R}}^{*} e'$ and $e_2 \rightarrow_{\mathbb{R}}^{*} e'$.*

3.3 Global Error Semantics

To define the global error semantics $\rightarrow_{\mathbb{E}}$, we first introduce the domain $\mathbb{E} = \mathbb{F} \times \mathbb{R}$. Intuitively, in a value $(x, \mu) \in \mathbb{E}$, μ measures the distance between the floating-point result of a computation x and the exact result. The elements of $\mathbb{E}$ are ordered by $(x_1, \mu_1) \prec (x_2, \mu_2) \iff \mu_1 \leq \mu_2$.

Formally, a value v is denoted by a pair (x, μ) where $x \in \mathbb{F}$ denotes the floating-point number used by the computer and $\mu \in \mathbb{R}$ denotes the exact error attached to x. For example, in simple precision, the real number $\frac{1}{3}$ is represented by the value $x = (\uparrow_{\circ} (\frac{1}{3}), \downarrow_{\circ} (\frac{1}{3})) = (0.333333, (\frac{1}{3} - 0.333333))$. The semantics interprets a constant c by $(\uparrow_{\circ} (c), \downarrow_{\circ} (c))$ and, for $v_1 = (x_1, \mu_1)$ and $v_2 = (x_2, \mu_2)$, the operations are defined by:

$$v_1 +_{\mathbb{E}} v_2 = (\uparrow_{\circ} (x_1 +_{\mathbb{R}} x_2), [\mu_1 + \mu_2 + \downarrow_{\circ} (x_1 +_{\mathbb{R}} x_2)]) \,, \tag{9}$$

$$v_1 \times_{\mathbb{E}} v_2 = (\uparrow_\circ (x_1 \times_{\mathbb{R}} x_2), [\mu_1 x_2 +_{\mathbb{R}} \mu_2 x_1 +_{\mathbb{R}} \mu_1\mu_2 +_{\mathbb{R}} \downarrow_\circ (x_1 \times_{\mathbb{R}} x_2)]) \,. \quad (10)$$

The global semantics $\rightarrow_{\mathbb{E}}$ is defined by the reduction rules of equations (4) to (8) of Figure 2 and by the domain $\mathbb{E}$ for the values. The operators $\oplus$ and $\otimes$ are the addition $+_{\mathbb{E}}$ and the product $\times_{\mathbb{E}}$.

Similarly to the semantics $\rightarrow_{\mathbb{R}}$ of Section 3.2, $\rightarrow_{\mathbb{E}}$ is non-deterministic since it also uses the rule of Equation (8) based on the syntactic relation $\equiv$. However, in $\mathbb{E}$, the operations are neither associative nor distributive and the reduction paths no longer are confluent.

Remark 2. *In general, for an arithmetic expression e, there exist reduction steps $e \rightarrow_{\mathbb{E}} e_1$ and $e \rightarrow_{\mathbb{E}} e_2$ such that there exists no expression e' such that $e_1 \rightarrow^*_{\mathbb{E}} e'$ and $e_2 \rightarrow^*_{\mathbb{E}} e'$.*

Nonetheless, the arithmetic $\mathbb{E}$ provides a way to compare the different execution paths of $\rightarrow_{\mathbb{E}}$ using the error measure μ attached to each value. We may consider a path $e \rightarrow^*_{\mathbb{E}} v_1$ is *better* than another path $e \rightarrow^*_{\mathbb{E}} v_2$ if $v_1 \prec v_2$. The code transformation introduced in the following sections consists of building a new arithmetic expression from the minimal trace corresponding to the evaluation of an expression e. But because there are possibly an exponential number of traces corresponding to the evaluation of e, we first merge some of them into abstract traces. The transformation is then based on the minimal abstract trace.

4 Abstract Semantics

The abstract semantics $\xrightarrow{A}_k$, introduced in Section 4.2, relies on the non-standard semantics $\longrightarrow$ of Section 4.1. In Section 4.3, we prove the correctness of the abstraction.

4.1 Non-standard Semantics

Basically, the non-standard semantics records, during a computation, how each intermediary result (sub-expression reduced to a value) was obtained. A label $\ell \in \mathcal{L}$ is attached to each value occurring in the expressions and we use two environments: The function $\rho : \mathcal{L} \rightarrow$Expr maps any label ℓ to the expression e whose evaluation has lead to v^ℓ. The environment σ : Expr $\rightarrow \mathbb{E}$ maps expressions to the result of their evaluation in the domain $\mathbb{E}$. We let Env_ρ and Env_σ denote the sets of such environments. This information is useful in the abstract semantics of Section 4.2.

Initially, a unique label is attached to each value occurring in an expression and a fresh label is associated to the result of each operation. For example, assuming that initially $\rho(\ell_1) = 1^{\ell_1}$, $\rho(\ell_2) = 2^{\ell_2}$ and $\rho(\ell_3) = 3^{\ell_3}$, the expression $(1^{\ell_1} + (2^{\ell_2} + 3^{\ell_3}))$ is evaluated as follows in the non-standard semantics:

$$\langle \rho, \sigma, (1^{\ell_1} + (2^{\ell_2} + 3^{\ell_3}))\rangle \rightarrow \langle \rho', \sigma', 1^{\ell_1} + 5^{\ell_4}\rangle \rightarrow \langle \rho'', \sigma'', 6^{\ell_5}\rangle$$

$$\frac{v = v_1 +_{\mathbb{E}} v_2 \quad \ell \notin \mathrm{Dom}(\rho)}{\langle \rho, \sigma, v_0^{\ell_0} + v_1^{\ell_1} \rangle \longrightarrow \langle \rho[\ell \mapsto \rho(\ell_1) + \rho(\ell_2)], \sigma[\rho(\ell_1) + \rho(\ell_2) \mapsto v], v^{\ell} \rangle} \tag{11}$$

$$\frac{v = v_1 \times_{\mathbb{E}} v_2 \quad \ell \notin \mathrm{Dom}(\rho)}{\langle \rho, \sigma, v_0^{\ell_0} \times v_1^{\ell_1} \rangle \longrightarrow \langle \rho[\ell \mapsto \rho(v_1^{\ell_1}) \times \rho(v_2^{\ell_2})], \sigma[\rho(\ell_1) \times \rho(\ell_2) \mapsto v], v^{\ell} \rangle} \tag{12}$$

$$\frac{\langle \rho, \sigma, e_0 \rangle \longrightarrow \langle \rho', \sigma', e_2 \rangle}{\langle \rho, \sigma, e_0 + e_1 \rangle \longrightarrow \langle \rho', \sigma', e_2 + e_1 \rangle} \tag{13}$$

$$\frac{\langle \rho, \sigma, e_0 \rangle \longrightarrow \langle \rho', \sigma', e_2 \rangle}{\langle \rho, \sigma, e_0 \times e_1 \rangle \longrightarrow \langle \rho', \sigma', e_2 \times e_1 \rangle} \tag{14}$$

$$\frac{e \equiv e_1 \qquad \langle \rho, \sigma, e_1 \rangle \longrightarrow \langle \rho', \sigma', e_2 \rangle \qquad e_2 \equiv e_3}{\langle \rho, \sigma, e_0 \rangle \longrightarrow \langle \rho', \sigma', e_3 \rangle} \tag{15}$$

Fig. 3. The non-standard semantics

where $\rho' = \rho[\ell_4 \mapsto 2^{\ell_2} + 3^{\ell_3}]$, $\sigma' = \sigma[2^{\ell_2} + 3^{\ell_3} \mapsto 5]$, $\rho'' = \rho'[\ell_5 \mapsto 1^{\ell_1} + (2^{\ell_2} + 3^{\ell_3})]$ and $s'' = \sigma'[1^{\ell_1} + (2^{\ell_2} + 3^{\ell_3}) \mapsto 6]$. In this example, for the sake of simplicity, values are integers instead of values of $\mathbb{E}$.

The non-standard semantics is given in Figure 3. We assume that, initially, $\rho(\ell) = v^{\ell}$ for any value v^{ℓ} occurring in the expression. Equations (11) and (12) respectively perform an addition and a product in $\mathbb{E}$. A new label ℓ is assigned to the result v of the operation and the environment ρ is extended in order to relate ℓ to the expression which has been evaluated. Similarly, σ is extended in order to record the result of the evaluation of the expression. The other rules only differ from the rules of the concrete semantics in that they propagate the environments ρ and σ.

4.2 Abstract Semantics

In order to decrease the size of the non-standard semantics, the abstract semantics merges traces in which sub-expressions have been evaluated approximatively in the same way. More precisely, instead of the environments ρ and σ, we use abstract environments $\rho^\sharp$ mapping labels to abstract expressions of limited height and abstract environments $\sigma^\sharp$ mapping expressions of limited height to unions of values. Next, we merge the paths in which sub-expressions have been evaluated almost in the same way, i.e. by the same abstract expressions.

From a formal point of view, the set $\mathrm{Expr}_k^\sharp$ of abstract expressions of height at most k is recursively defined by:

$$\begin{array}{l} \eta_0 ::= v^{\sharp\ell} \mid \top_\eta \\ \eta_k ::= \eta_{k-1} \mid \eta_{k-1} + \eta_{k-1} \mid \eta_{k-1} \times \eta_{k-1}. \end{array} \tag{16}$$

The values occurring in the abstract expressions belong to the abstract domain $\mathbb{E}^\sharp$. Let $\wp(X)$ denote the powerset of X. Abstract and concrete floating-point numbers with errors are related by the Galois connection

$$\langle \wp(\mathbb{E}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathbb{E}^\sharp, \sqsubseteq_{\mathbb{E}}^\sharp \rangle.$$

$$\frac{v^\sharp = \bigcup \sigma^\sharp(\eta_1) +^\sharp_{\mathbb{E}} \sigma^\sharp(\eta_2) \quad E = \bigcup \ulcorner \eta_1 + \eta_2 \urcorner^k \quad \sigma^{\sharp\prime} = \sigma^\sharp \bigodot [\eta \mapsto \sigma^\sharp(\eta) \cup \nu]}{\langle \rho^\sharp, \sigma^\sharp, v_0^{\ell_0} + v_1^{\ell_1} \rangle \xrightarrow{\ell=\ell_1+\ell_2}_k \langle \rho^\sharp[\ell \mapsto \rho^\sharp(\ell) \cup E], \sigma^{\sharp\prime}, v^\ell \rangle} \quad (17)$$

where the premises are $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$; E ranges over $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$; and $\sigma^{\sharp\prime}$ ranges over $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$, $\eta = \ulcorner \eta_1 + \eta_2 \urcorner^k$, $\nu = \sigma^\sharp(\eta_1) + \sigma^\sharp(\eta_2)$.

$$\frac{v^\sharp = \bigcup \sigma^\sharp(\eta_1) \times^\sharp_{\mathbb{E}} \sigma^\sharp(\eta_2) \quad E = \bigcup \ulcorner \eta_1 \times \eta_2 \urcorner^k \quad \sigma^{\sharp\prime} = \sigma^\sharp \bigodot [\eta \mapsto \sigma^\sharp(\eta) \cup \nu]}{\langle \rho^\sharp, \sigma^\sharp, v_0^{\ell_0} \times v_1^{\ell_1} \rangle \xrightarrow{\ell=\ell_1\times\ell_2}_k \langle \rho^\sharp[\ell \mapsto \rho^\sharp(\ell) \cup E], \sigma^{\sharp\prime}, v^\ell \rangle} \quad (18)$$

where the premises are $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$; E ranges over $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$; and $\sigma^{\sharp\prime}$ ranges over $\eta_1 \in \rho^\sharp(\ell_1)$, $\eta_2 \in \rho^\sharp(\ell_2)$, $\eta = \ulcorner \eta_1 \times \eta_2 \urcorner^k$, $\nu = \sigma^\sharp(\eta_1) \times \sigma^\sharp(\eta_2)$.

$$\frac{\langle \rho^\sharp, \sigma^\sharp, e_0 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2 \rangle}{\langle \rho^\sharp, \sigma^\sharp, e_0 + e_1 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2 + e_1 \rangle} \quad (19)$$

$$\frac{\langle \rho^\sharp, \sigma^\sharp, e_0 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2 \rangle}{\langle \rho^\sharp, \sigma^\sharp, e_0 \times e_1 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2 \times e_1 \rangle} \quad (20)$$

$$\frac{e \equiv_k e_1 \qquad \langle \rho^\sharp, \sigma^\sharp, e_1 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2 \rangle \qquad e_2 \equiv_k e_3}{\langle \rho^\sharp, \sigma^\sharp, e_0 \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_3 \rangle} \quad (21)$$

Fig. 4. The abstract semantics

This connection abstracts sets of values of $\mathbb{E}$ by intervals in a componentwise way. The partial order $\sqsubseteq^\sharp_{\mathbb{E}}$ is the componentwise inclusion order on intervals.

An expression e of arbitrary height can be abstracted by $\eta \in \mathrm{Expr}^\sharp_k$ by means of the operator $\ulcorner e \urcorner^k$ recursively defined as follows:

$$\begin{array}{rll}
\ulcorner v^\ell \urcorner^k &= v^\ell & k \geq 0 \\
\ulcorner \top_\eta \urcorner^k &= \top_\eta & k \geq 0 \\
\ulcorner e_1 + e_2 \urcorner^0 &= \top_\eta & \\
\ulcorner e_1 \times e_2 \urcorner^0 &= \top_\eta & \\
\ulcorner e_1 + e_2 \urcorner^k &= \ulcorner e_1 \urcorner^{k-1} + \ulcorner e_2 \urcorner^{k-1} & k \geq 1 \\
\ulcorner e_1 \times e_2 \urcorner^k &= \ulcorner e_1 \urcorner^{k-1} \times \ulcorner e_2 \urcorner^{k-1} & k \geq 1
\end{array}$$

Intuitively, $\ulcorner e \urcorner^k$ replaces in e all the nodes of height k which are not values by $\top_\eta$. The function $\ulcorner . \urcorner^k$ is indifferently applied to expressions $e \in \mathrm{Expr}$ or abstract expressions $\eta \in \mathrm{Expr}^\sharp_{k'}$ for any integer k'.

The abstract semantics, given in Figure 4, uses reduction rules of the form $\langle \rho^\sharp, \sigma^\sharp, e \rangle \xrightarrow{A}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e' \rangle$. The symbol k is a parameter of the semantics and A is an action indicating which operation is actually performed by the transition. Actions are used to build a new arithmetic expression from a trace and are detailed in Section 5. The environment $\rho^\sharp : \mathcal{L} \to \wp(\mathrm{Expr}^\sharp_k)$ maps labels to sets of abstract expressions. The environment $\sigma^\sharp : \mathrm{Expr}^\sharp_k \to \mathbb{E}^\sharp$ maps abstract

expressions to abstract values. The symbols $\mathrm{Env}^\sharp_\rho$ and $\mathrm{Env}^\sharp_\sigma$ denote the sets of such environments. Intuitively, $\eta \in \mathrm{Expr}^\sharp_k$ abstracts a set S of expressions and $\sigma^\sharp$ relates η to an abstract value containing all the possible values resulting from the evaluation of $e \in S$.

The expression $\sigma^\sharp[\eta \mapsto v^\sharp]$ denotes the environment $\sigma^\sharp$ extended by $\sigma^\sharp(\eta) = v^\sharp$ and

$$\sigma^\sharp \bigodot_{\eta \in S,\ v^\sharp = f(\eta)} [\eta \mapsto v^\sharp]$$

is a shortcut for $\sigma^\sharp[\eta_1 \mapsto f(\eta_1)][\eta_2 \mapsto f(\eta_2)] \ldots [\eta_n \mapsto f(\eta_n)]$, for all $\eta_i, 1 \leq i \leq n$, such that $\eta_i \in S$.

In Figure 4, Equation (17) and Equation (18) are used for the addition and for the product of two values, respectively. Let us assume that $v_0^{\ell_0} + v_1^{\ell_1}$ is the current expression. The values v_1 and v_2 result from the evaluation of some expressions $\eta_1 \in \rho^\sharp(\ell_1)$ and $\eta_2 \in \rho^\sharp(\ell_2)$ (assuming that, initially, $\rho^\sharp(\ell) = v^{\sharp\ell}$ for any value $v^{\sharp\ell}$ occurring in the expression.) So, $v^\sharp = \sigma^\sharp(\eta_1) + \sigma^\sharp(\eta_2)$. In Equation (17), the result $v^\sharp$ of the addition is obtained by joining the sums of all the possible operands in $\sigma^\sharp(\eta_1)$ and in $\sigma^\sharp(\eta_2)$, for all possible abstract expressions $\eta_1 \in \rho^\sharp(\ell_1)$ and $\eta_2 \in \rho^\sharp(\ell_2)$. Next, a fresh label ℓ is attached to $v^\sharp$ and $\rho^\sharp$ is modified by assigning to ℓ the set E of abstract expressions which have possibly been used to compute $v^\sharp$. Finally, $\sigma^\sharp$ is updated: it is extended by assignments $[\eta \mapsto \sigma^\sharp(\eta) \cup \nu]$ where η is one of the possible expressions used to compute v and ν is the corresponding abstract value.

Equation (18) is similar to Equation (17) and equations (19) and (20) present no difficulty. Equation (21) is similar to equations (8) and (15): it introduces non-determinism in the semantics by means of a syntactic equivalence relation but now we use a new relation $\equiv_k$ instead of the previous relation $\equiv$.

Definition 3. *Let $\sim_k \subseteq Expr \times Expr$ be the equivalence relation defined by:*

$$e \sim_k e' \iff \ulcorner e \urcorner^k = \ulcorner e' \urcorner^k.$$

Then $\equiv_k \subseteq Expr \times Expr$ is the quotient relation $\equiv\ /\ \sim_k$.

Let us remark that $\equiv_k$ is coarser than $\equiv$ (which means that $\equiv \subseteq \equiv_k$) and that, while the $\equiv$-class $Cl_\equiv(e)$ of an expression e contains all the expressions generated by the rules (i) to (vii) of Figure 2, the $\equiv_k$-class $Cl_{\equiv_k}(e)$ contains only one element of each $\sim_k$ class among the $\equiv$-equivalent elements.

At each step, our concrete and abstract semantics generate one new path per element of the equivalence class of the current expression. As $\equiv_k$ is coarser than $\equiv$, the number of paths of the semantics based on $\equiv_k$ is smaller than the number of paths of the semantics based on $\equiv$.

Property 4. *Let e be an expression of size n.*

(i) In the worst case, the $\equiv$-class of e contains $O(exp(n))$ elements.
(ii) In the worst case, the $\equiv_k$-class of e contains $O(n^k)$ elements.

This property states that the number of expressions $\equiv_k$-equivalent to a given expression e is a polynomial of degree k, in the size of e. A worst case consists of taking a sequence of sums $x_1 + x_2 + \ldots + x_n$ which, by associativity, can be evaluated in $n!$ ways using $\equiv$ and in n^k ways using $\equiv_k$, where k is a user-defined parameter of the semantics.

4.3 Correctness of the Abstract Semantics

In this section, we show that the abstract semantics of Section 4.2 is a correct abstraction of the non-standard semantics of Section 4.1. First, we relate the environments used in the non-standard semantics and in the abstract semantics by the Galois connections

$$\langle \wp(\mathrm{Env}_\rho), \subseteq \rangle \underset{\alpha_k^\rho}{\overset{\gamma_k^\rho}{\leftrightarrows}} \langle \mathrm{Env}^\sharp_{\rho,k}, \sqsubseteq_\rho \rangle \tag{22}$$

and

$$\langle \wp(\mathrm{Env}_\sigma), \subseteq \rangle \underset{\alpha_k^\sigma}{\overset{\gamma_k^\sigma}{\leftrightarrows}} \langle \mathrm{Env}^\sharp_{\sigma,k}, \sqsubseteq_\sigma \rangle. \tag{23}$$

The partial order as well as abstraction and concretization functions for the first kind of environments are defined by

$$\rho_1^\sharp \sqsubseteq_\rho \rho_2^\sharp \iff \forall \ell \in \mathrm{Dom}(\rho_1^\sharp),\ \rho_1^\sharp(\ell) \subseteq \rho_2^\sharp(\ell), \tag{24}$$

$$\alpha_k^\rho(R) = \rho^\sharp \ :\ \forall \ell \in \mathcal{L},\ \rho^\sharp(\ell) = \cup_{\rho \in R} \ulcorner \rho(\ell) \urcorner^k, \tag{25}$$

$$\gamma_k^\rho(\rho^\sharp) = \{\rho \in \mathrm{Env}_\rho \ :\ \forall \ell \in \mathcal{L},\ \ulcorner \rho(\ell) \urcorner^k \in \rho^\sharp(\ell)\}. \tag{26}$$

The environment $\rho_1^\sharp$ is smaller than $\rho_2^\sharp$ if, for any label ℓ, the set $\rho_1^\sharp(\ell)$ is a subset of $\rho_2^\sharp(\ell)$. The abstraction $\alpha_k^\rho(R)$ of a set $R = \{\rho_1, \rho_2, \ldots, \rho_n\}$ of environments is the abstract environment $\rho^\sharp$ which maps any label ℓ to the set of abstract expressions $\ulcorner e \urcorner^k$ such that $\rho_i(\ell) = e$ for some $1 \leq i \leq n$. Conversely, γ_k^ρ is the set of environments ρ which map ℓ to an expression e such that $\ulcorner e \urcorner^k = \rho^\sharp(\ell)$. Similarly, we have for the second kind of environments

$$\sigma_1^\sharp \sqsubseteq_\sigma \sigma_2^\sharp \iff \forall \eta \in \mathrm{Dom}(\sigma_1^\sharp),\ \sigma_1^\sharp(\eta) \sqsubseteq_{\mathbb{E}}^\sharp \sigma_2^\sharp(\eta), \tag{27}$$

$$\alpha_k^\sigma(S) = \sigma^\sharp \ :\ \forall \eta \in \mathrm{Expr}_k^\sharp,\ \sigma^\sharp(\eta) = \alpha(\{\sigma(\eta),\ \sigma \in S\}), \tag{28}$$

$$\gamma_k^\sigma(\sigma^\sharp) = \{\sigma \in \mathrm{Env}_\sigma \ :\ \forall e \in \mathrm{Expr},\ \sigma(e) \in \gamma(\sigma^\sharp(\ulcorner e \urcorner^k))\}. \tag{29}$$

The environment $\sigma_1^\sharp$ is smaller than $\sigma_2^\sharp$ if $\sigma_1^\sharp$ maps any abstract expression η to an abstract value smaller than $\sigma_2^\sharp$. The abstraction α_k^σ and concretization γ_k^σ are based on the Galois connection introduced in Section 4.2 to relate concrete and abstract values.

Let $\langle \rho, \sigma, e\rangle \longrightarrow^n \langle \rho', \sigma', v\rangle$ and $\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}{}^n_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle$ denote sequences of reduction steps of length n in the non-standard and abstract semantics, yielding final values v and $v^\sharp$, respectively. The following property holds.

Property 5. *If* $\langle \rho, \sigma, e\rangle \longrightarrow^n \langle \rho', \sigma', v\rangle$ *and if* $\alpha^\rho_k(\rho) \sqsubseteq_\rho \rho^\sharp$ *and* $\alpha^\sigma_k(\sigma) \sqsubseteq_\sigma \sigma^\sharp$ *then* $\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}{}^n_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle$ *such that* $v \in \gamma(v^\sharp)$, $\alpha^\rho_k(\rho') \sqsubseteq_\rho \rho^{\sharp\prime}$ *and* $\alpha^\sigma_k(\sigma') \sqsubseteq_\sigma \sigma^{\sharp\prime}$.

Property 5 states that for any path of length n, in the non-standard semantics which leads to a value v, there exists a path of the abstract semantics of length n which leads to a value $v^\sharp$ such that $v \in \gamma(v^\sharp)$.

Proof

The proof is by induction on the length n of the reduction sequence. If $n = 1$ then $e = v_1^{\ell_1} + v_2^{\ell_2}$ or $e = v_1^{\ell_1} \times v_2^{\ell_2}$. Let us assume that $e = v_1^{\ell_1} + v_2^{\ell_2}$ (the case $e = v_1^{\ell_1} \times v_2^{\ell_2}$ is similar). Let $v = v_1 + v_2$. In the non-standard semantics we have:

$$\langle \rho, \sigma, e\rangle \longrightarrow \langle \rho[\ell \mapsto \rho(\ell_1) + \rho(\ell_2)], \sigma[\rho(\ell_1) + \rho(\ell_2) \mapsto v], v^\ell\rangle$$

In the abstract semantics we have $\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}{}^n_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle$ with:

$$v^\sharp = \bigcup_{\substack{\eta_1 \in \rho^\sharp(\ell_1)\\ \eta_2 \in \rho^\sharp(\ell_2)}} \sigma^\sharp(\eta_1) + \sigma^\sharp(\eta_2)$$

As $\rho(\ell_1) = v_1^{\ell_1}$, $\rho(\ell_2) = v_2^{\ell_2}$, since by hypothesis, $\alpha^\rho_k(\rho) \sqsubseteq_\rho \rho^\sharp$ and $\alpha^\sigma_k(\sigma) \sqsubseteq_\sigma \sigma^\sharp$, and also because in a Galois connection, $\gamma \circ \alpha$ is extensive ($R \subseteq \gamma^\rho_k(\alpha^\rho_k(R))$ and $S \subseteq \gamma^\sigma_k(\alpha^\sigma_k(S))$), we thus have $v_1 \in \gamma(\sigma^\sharp(\rho^\sharp(\ell_1)))$ and $v_2 \in \gamma(\sigma^\sharp(\rho^\sharp(\ell_2)))$. Consequently, $v \in \gamma(v^\sharp)$.

The proof for $n = 1$ is completed without difficulty by showing that $\alpha^\rho_k(\rho') \sqsubseteq_\rho \rho^{\sharp\prime}$ and $\alpha^\sigma_k(\sigma') \sqsubseteq_\sigma \sigma^{\sharp\prime}$ with $\rho' = \rho[\ell \mapsto \rho(\ell_1)+\rho(\ell_2)]$ and $\sigma' = \sigma[\rho(\ell_1)+\rho(\ell_2) \mapsto v]$.

Now, we assume that the property holds for any $m \leq n$ and we consider a sequence of length $n + 1$. We distinguish two cases:

- Rules of Equation (13) and Equation (14): if $\frac{\langle \rho,\sigma,e_0\rangle \longrightarrow \langle \rho',\sigma',e_2\rangle}{\langle \rho,\sigma,e_0+e_1\rangle \longrightarrow \langle \rho',\sigma',e_2+e_1\rangle}$ then $\frac{\langle \rho^\sharp,\sigma^\sharp,e_0\rangle \xrightarrow{A}{}_k \langle \rho^{\sharp\prime},\sigma^{\sharp\prime},e_2\rangle}{\langle \rho^\sharp,\sigma^\sharp,e_0+e_1\rangle \xrightarrow{A}{}_k \langle \rho^{\sharp\prime},\sigma^{\sharp\prime},e_2+e_1\rangle}$. By our induction hypothesis, $\alpha^\rho_k(\rho') \sqsubseteq_\rho \rho^{\sharp\prime}$ and $\alpha^\sigma_k(\sigma') \sqsubseteq_\sigma \sigma^{\sharp\prime}$. Now, $\langle \rho', \sigma', e'\rangle \longrightarrow^n \langle \rho'', \sigma'', v\rangle$ and we may apply again our induction hypothesis.
- Rule of Equation (15): let us assume that $\frac{e \equiv e_1 \qquad \langle \rho,\sigma,e_1\rangle \longrightarrow \langle \rho',\sigma',e_2\rangle \qquad e_2 \equiv e_3}{\langle \rho,\sigma,e_0\rangle \longrightarrow \langle \rho',\sigma',e_3\rangle}$. Then, since, by definition of $\equiv_k$, $\equiv \subseteq \equiv_k$, $e \equiv e_1 \Rightarrow e \equiv_k e_1$ and $e_2 \equiv e_3 \Rightarrow e_2 \equiv_k e_3$. So, in the abstract semantics we have:

$$\frac{e \equiv_k e_1 \qquad \langle \rho^\sharp, \sigma^\sharp, e_1\rangle \xrightarrow{A}{}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_2\rangle \qquad e_2 \equiv_k e_3}{\langle \rho^\sharp, \sigma^\sharp, e_0\rangle \xrightarrow{A}{}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e_3\rangle}$$

  Then we can complete the proof, by induction, in the same way as in the previous case. □

5 Semantics Transformation

The concrete semantics of an arithmetic expression is the floating-point semantics $\rightarrow_{\mathbb{F}}$ defined in Section 3.1. Indeed, this is the only semantics which indicates how an expression is actually evaluated by a computer. Given an expression e and its (unique) execution trace $t = e \rightarrow_{\mathbb{F}}^{*} v$, the semantics transformation has to generate a new trace $t' = e' \rightarrow_{\mathbb{F}}^{*} v'$ such that t and t' are equal at some observational level. This is performed in Section 5.1 by using the information provided by the abstract semantics $\xrightarrow{A}_k$. In Section 5.2, we prove that $e \rightarrow_{\mathbb{R}}^{*} v''$ and $e' \rightarrow_{\mathbb{R}}^{*} v''$ for the same value v'', where $\rightarrow_{\mathbb{R}}$ is the semantics introduced in Section 3.2.

5.1 Semantics Transformation

Because the abstract semantics $\xrightarrow{A}_k$ of an expression e, as defined in Section 4.2, is non-deterministic, the abstract interpretation of e consists of a set of traces. The semantics transformation τ_k is based on the trace $e \xrightarrow{A}_k^{*} v^\sharp$ which optimizes the quality of the evaluation: recall, from Section 3.3, that, in the global error based semantics, any value is a pair $(x, \mu) \in \mathbb{E}$ where x is a computer representable value and μ a measure of the quality of x. Recall also that $(x_1, \mu_1) \prec (x_2, \mu_2) \iff \mu_1 \leq \mu_2$. Let $\mu_1^\sharp = [\underline{\mu_1}, \overline{\mu_1}]$ and $\mu_2^\sharp = [\underline{\mu_2}, \overline{\mu_2}]$. The corresponding order in $\mathbb{E}^\sharp$ is:

$$(x_1^\sharp, \mu_1^\sharp) \prec^\sharp (x_2^\sharp, \mu_2^\sharp) \iff \max(|\underline{\mu_1}|, |\overline{\mu_1}|) \leq \max(|\underline{\mu_2}|, |\overline{\mu_2}|) \tag{30}$$

In $\prec^\sharp$, $v_1^\sharp$ is more precise than $v_2^\sharp$ if, in absolute value, the maximal error on $v_1^\sharp$ is less than the maximal error on $v_2^\sharp$.

The transformation τ_k is based on the minimal abstract trace $e \xrightarrow{A}_k^{*} v^\sharp$, i.e. the trace which yields the minimal value $v^\sharp$, in the sense of $\prec^\sharp$. Remark that, since $\xrightarrow{A}_k$ uses abstract values of $\mathbb{E}^\sharp$, the transformation τ_k minimizes the worst error μ which may occurs during an evaluation. Therefore, τ_k minimizes the precision lost which may arise during an evaluation in the worst case, that is for the most pessimistic combination of data.

Because the semantics $\xrightarrow{A}_k$ allows more steps than the semantics $\rightarrow_{\mathbb{F}}$ (in $\rightarrow_{\mathbb{F}}$ an expression may not be transformed by $\equiv_k$), we cannot directly transform a trace of $\xrightarrow{A}_k$ into a trace of $\rightarrow_{\mathbb{F}}$: we first have to rebuild the totally parsed expression which has actually been evaluated by $\xrightarrow{A}_k$. This is achieved by using the actions A appearing in the transitions of the abstract semantics and which collect the operations actually performed along a trace.

Actions are expressions of the form $\ell = \ell_1 + \ell_2$ or $\ell = \ell_1 \times \ell_2$, where ℓ, ℓ_1 and ℓ_2 are labels belonging to $\mathcal{L}$. An action $\ell = \ell_1 + \ell_2$ indicates that the value of label ℓ is the addition of the expressions of labels ℓ_1 and ℓ_2.

The expression generation function $\mathbf{P}$ is defined in Figure 5. $\mathbf{P}$ takes a trace, an environment $\iota : \mathcal{L} \rightarrow$Expr and computes a new environment ι'. For a trace

$$\mathbf{P}\left(\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{\ell=\ell_1+\ell_2}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e'\rangle, \iota\right) = \iota[\ell \mapsto \iota(\ell_1) + \iota(\ell_2)] \tag{31}$$

$$\mathbf{P}\left(\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{\ell=\ell_1\times\ell_2}_k \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, e'\rangle, \iota\right) = \iota[\ell \mapsto \ell_1 \times \ell_2] \tag{32}$$

$$\mathbf{P}\left(\langle \rho^\sharp, \sigma^\sharp, v^{\sharp\ell}\rangle, \iota\right) = \iota(\ell) \tag{33}$$

$$\mathbf{P}\left(s_1 \xrightarrow{A}_k s_2 \xrightarrow{A}_k \ldots s_n, \iota\right) = \mathbf{P}\left(s_2 \xrightarrow{A}_k \ldots s_n, \mathbf{P}(s_1 \xrightarrow{A}_k s_2)\right) \tag{34}$$

Fig. 5. Generation of the new expression

$t^\sharp = \langle \rho^\sharp, s^\sharp, e\rangle \xrightarrow{A}_k^* \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle$, initially assuming that $\iota(\ell) = v$ for any value v^ℓ occurring in the source expression e, $\mathbf{P}(t^\sharp, \iota) = \iota'(\ell)$, where $\iota'(\ell)$ is the expression actually evaluated by $t^\sharp$.

Let e be an arithmetic expression and let $\mathcal{T}_k^\sharp(e)$ denote the set of evaluation traces in the abstract semantics $\xrightarrow{A}_k$ of e, i.e. $\mathcal{T}_k^\sharp(e) = \{\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}_k^* \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle\}$. The minimal trace of $\mathcal{T}_k^\sharp(e)$ is

$$\min_{\prec^\sharp} \mathcal{T}_k^\sharp(e) = \langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}_k^* \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle,$$

where $v^\sharp \prec^\sharp v^{\sharp\prime}$ whenever $\langle \rho^\sharp, \sigma^\sharp, e\rangle \xrightarrow{A}_k^* \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^{\sharp\prime}\rangle \in \mathcal{T}_k^\sharp(e)$.

The transformation is defined as follows:

Definition 6. *Let e be an arithmetic expression. The semantics transformation τ_k of $e \to_{\mathbb{F}} v$ is defined by*

$$\tau_k\left(e \to_{\mathbb{F}} v, \mathcal{T}_k^\sharp(e)\right) = \mathbf{P}\left(min_{\prec^\sharp} \mathcal{T}_k^\sharp(e)\right) \to_{\mathbb{F}} v'. \tag{35}$$

By Definition 6, the transformed trace is the evaluation trace in the floating-point arithmetic based semantics of the expression $\mathbf{P}(e)$ generated from the minimal trace $e \xrightarrow{A}_k^* v^\sharp = \min_{\prec^\sharp} \mathcal{T}_k^\sharp(e)$.

5.2 Correctness of the Transformation

In order to prove the correctness of the transformation, we show that, at an observational level [6], the semantics of the original expression e and the semantics of the transformed expression e_t are equal. Our observation consists of showing that e and e_t compute the same thing in the exact arithmetic of real numbers.

Let $\alpha_{\mathcal{O}}$ be an observational abstraction $\alpha_{\mathcal{O}} : \mathbb{E} \to \mathbb{R}$ which transforms a floating-point number with errors into a real number, i.e. $\alpha_{\mathcal{O}}(x, \mu) = x + \mu$. We first introduce a lemma concerning the non-standard semantics.

Lemma 7. *Let e be an arithmetic expression and let $\langle \rho, \sigma, e\rangle \longrightarrow^* \langle \rho', \sigma', v_1\rangle$ and $\langle \rho, \sigma, e\rangle \longrightarrow^* \langle \rho'', \sigma'', v_2\rangle$ be two paths of the non-standard semantics. Then $\alpha_{\mathcal{O}}(v_1) = \alpha_{\mathcal{O}}(v_2)$.*

Lemma 7 stems from the fact that, in $\mathbb{E}$, the errors are exactly computed. So, from the perspective of $\alpha_{\mathcal{O}}$, the traces of $\rightarrow_{\mathbb{E}}$ are identical to the traces of $\rightarrow_{\mathbb{R}}$.

Lemma 8. *Let* $e_t = \mathbf{P}\left(t^\sharp, \iota\right)$ *for some trace* $\langle \rho^\sharp, s^\sharp, e\rangle \xrightarrow{A}{}^{*}_{k} \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle \in \mathcal{T}^\sharp_k(e)$. *Then* $e \equiv e_t$.

As a consequence, in the non-standard semantics e and e_t lead to observationally equivalent values. By Lemma 7, if $\langle \rho, \sigma, e\rangle \longrightarrow^* \langle \rho', \sigma', v\rangle$ then, by the rule of Equation (15), $\langle \rho, \sigma, e_t\rangle \longrightarrow^* \langle \rho'', \sigma'', v\rangle$. Using Lemma 7 and Lemma 8, we have:

Property 9. *Let* e *be an arithmetic expression, let* $t = e \rightarrow^*_{\mathbb{F}} v$ *be the concrete evaluation trace of* e, *and let* $t^\sharp = \langle \rho^\sharp, s^\sharp, e\rangle \xrightarrow{A}{}^{*}_{k} \langle \rho^{\sharp\prime}, \sigma^{\sharp\prime}, v^\sharp\rangle \in \mathcal{T}^\sharp_k(e)$. *We have:*

$$e \rightarrow^*_{\mathbb{R}} v_{\mathbb{R}} \iff \mathbf{P}(t^\sharp, \iota) \rightarrow^*_{\mathbb{R}} v_{\mathbb{R}} \tag{36}$$

In particular, this property holds for the minimal trace used in Equation (35), in the definition of τ_k.

6 Experimental Results

A prototype based on the abstract semantics of Section 4.2 and on the transformation of Section 5 has been implemented and, in this section, we present some experimental results.

As explained in Section 2, adding numbers of different magnitudes may lead to important precision loss by absorption. For example, in the IEEE754 simple-precision format, $1.0 + 5e^{-8} = 1.0$ while $1.0 + (2 \times 5e^{-8}) \neq 1.0$. We consider the expression:

$$e = a \times ((b + c) + d)$$

and the global abstract environment $\theta^\sharp$ such that:

$$a = [56789, 98765] \quad b = [0, 1] \quad c = [0, 5e^{-8}] \quad d = [0, 5e^{-8}] \tag{37}$$

Our prototype computes for this example (with $k = 2$):

```
(a*((b+c)+d)) -> ((a*b)+(a*(c+d)))
```

The sums are parsed in order to first add the smallest terms: this limits the absorption. Furthermore, the product is distributed and this avoids the multiplication of roundoff errors of the additions by a large value and, consequently, this also reduces the final error.

Using the domain $\mathbb{E}^\sharp$, which computes an over-approximation of the error attached to the result of a floating-point computation, our prototype also outputs a bound on the maximal error arising during the evaluation of an expression (for any concrete set of inputs in the intervals given in Equation (37)). The errors for the source and transformed expressions are:

- Error bound on `(a*((b+c)+d))`: [-1.5679E-2,1.5680E-2]
- Error bound on `((a*b)+(a*(c+d)))`: [-7.8125E-3,7.8126E-3]

The error on the transformed expression is approximatively half the error on the original expression.

Our second example concerns the sum $s = \sum_{i=0}^{4} x_i$, with $x_i = [2^i, 2^{i+1}]$. The results, for different values of k are given in the table below, where `a`, `b`, `c`, `d` and `e` stand for x_0, x_1, x_2, x_3 and x_4, respectively:

Case	Expression	Error bound
Source expression	`(((e+d)+c)+b)+a`	[-7.6293E-6,7.6294E-6]
$k = 1$	`(b+a)+(c+(e+d))`	[-5.9604E-6,5.9605E-6]
$k = 2$	`(c+(b+a))+(e+d)`	[-4.5299E-6,4.5300E-6]
$k = 3$	`(d+(c+(a+b)))+e`	[-3.5762E-6,3.5763E-6]

As the parameter k increases, the terms are more and more sorted, increasing the precision of the result. With $k = 3$, the error is guaranteed to be less than half the error on the original expression.

Another class of examples concerns the evaluation of polynomials. Again, anybody familiar with computer arithmetic knows that, in general, factorization improves the quality of the evaluation of a polynomial. In the abstract environment $\theta^\sharp$ in which an initial error has been attached to x: for $x = ([0, 2], [0, 0.0005])$, we obtain the following results:

Case	Expression	Error bound
Source expression	`x+(x*x)`	[-1.800074334E-3,1.001074437E-3]
$k = 2$	`(1.0+x)*x`	[-9.000069921E-4,1.010078437E-4]
Source expression	`(x*(x*x))+(x*x)`	[-1.802887642E-3,3.191200091E-3]
$k = 3$	`(x+1.0)*(x*x)`	[-1.818142851E-4,1.390014781E-3]
$k = 4$	`((1.0+x)*x)*x`	[-9.091078216E-5,1.100112212E-3]

Our last example concerns the expression $(a + b)^2$. If $b \ll a$, then we obtain a better precision by developing the remarkable identity. Using $a = [5, 10]$ and $b = [0, 0.001]$, our prototype outputs the following results.

Case	Expression	Error bound
Source expression	`(a+b)*(a+b)`	[-1.335239380E-5,1.335239381E-5]
$k = 2$	`((b*(a+b))+(a*b))+(a*a)`	[-7.631734013E-6,7.631734014E-6]
$k = 3$	`(((b*a)+(b*b))+(b*a))+(a*a)`	[-7.631722894E-6,7.631722895E-6]

With $k = 3$ the transformation consists of finding the remarkable identity. However, with $k = 2$, another formula which significantly improve the precision has already been found.

7 Perspectives

We believe that the new kind of program transformation introduced in this article can be improved and extended in many ways.

First of all, we aim at extending our methodology to full programming languages, with variables, loops and conditionals, instead of simple arithmetic expressions. We believe it is possible to rewrite computations defined among many lines of code. General code transformation techniques [10] could be used. For example, loop unfolding techniques can be used to improve the numerical precision of iterative computations. In addition, some statements may also introduce precision loss, like assignments when processor registers have more digits than

memory locations [11]. This last remark also makes us believe that our program transformation could be used on assembler codes, possibly at compile-time. We are confident in the feasibility of such transformations for large scale programs, static analyses for numerical precision having already been defined for general programming languages and being implemented in analyzers used in industrial contexts [8].

Another research direction concerns the abstract semantics. In this article, we have presented a simple abstract semantics which could be improved in many ways. For arithmetic expressions, more subtle abstractions could be defined, which more globally minimize the error on an evaluation path. The semantics of error series [13] could be useful in this context, but we believe that other approaches could also be successfully developed.

The relation $\equiv$, introduced in Section 2, identifies expressions which are equal in the reals. These laws enable us to rewrite expressions. However, the relation $\equiv$ is not unique and could be extended by many other laws. For example, some laws can be used to improve the precision of floating-point computations, like Sterbenz's theorem for subtraction [14]. Other laws can be found in [3,4,2].

Finally, other applications could be studied. For example, finite precision arithmetic is widely used in embedded systems. In order to implement a chain of operations, the programmer often works as follows: the size of the inputs (their number of digits) is know, and the result r of each elementary operation is stored in a new number large enough to represent exactly r. Obviously, the designer of an embedded system aims at limiting the sizes of the numbers and this strongly depends on how the formula is implemented. Yet other applications, like code obfuscation for arithmetic expressions without loss of precision, could also be developed, the framework of semantics program transformation having already been used in this context [7].

8 Conclusion

In this article, we have introduced a semantics-based program transformation for arithmetic expressions, in order to improve the quality of their implementation. This work is a first step towards the automatic improvement of large scale codes containing numerical computations. This research direction could find many applications, in the context of embedded softwares as well as for numerical codes. In addition, this program transformation can be used either as a source to source transformation or at compile-time, during the low-level code generation phase.

We believe that our method can be improved and extended in many directions and some issues have been discussed in Section 7. Meanwhile, the experimental results of Section 6 show that the transformation of simple arithmetic expressions, using a simple analysis, already yield interesting results.

We also believe that the framework of semantics program transformation [6] was very helpful to define our method, which would have been more difficult to design and prove at the syntactic level.

More generally, our approach relies on the assumption that, concerning numerical precision, a program can be viewed either as a model or as an implementation. More precisely, a formula occurring in a source code may be considered as the specification of what should be computed in the reals as well as a sequence of machine operations. We used the first point of view to generate a new sequence of operations. We believe that this approach may lead to many further developments in the domain of program transformation for numerical precision, independently of the techniques used in this article which represent our first attempt to automatically improve the accuracy of numerical programs.

References

1. ANSI/IEEE. IEEE Standard for Binary Floating-point Arithmetic, std 754-1985 edition (1985)
2. Bohlender, G., Walter, W., Kornerup, P., Matula, D.W.: Semantics for exact floating-point operations. In: Symp. on Computer Arithmetic, pp. 22–26 (1991)
3. Boldo, S., Daumas, M.: Properties of the subtraction valid for any floating point system. In: 7th International Workshop on Formal Methods for Industrial Critical Systems, pp. 137–149 (2002)
4. Boldo, S., Daumas, M.: Representable correcting terms for possibly underflowing floating point operations. In: Bajard, J.-C., Schulte, M. (eds.) 16th Symposium on Computer Arithmetic, pp. 79–86 (2003)
5. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points. In: Principles of Programming Languages 4, pp. 238–252. ACM Press, New York (1977)
6. Cousot, P., Cousot, R.: Systematic design of program transformation frameworks by abstract interpretation. In: Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, pp. 178–190. ACM Press, New York (2002)
7. Dalla Preda, M., Giacobazzi, R.: Control code obfuscation by abstract interpretation. In: International Conference on Software Engineering and Formal Methods, SEFM'05, pp. 301–310. IEEE Computer Society Press, Los Alamitos (2005)
8. Goubault, E., Martel, M., Putot, S.: Some future challenges in the validation of control systems. In: ERTS'06 (2006)
9. Goubault, E., Putot, S.: Static analysis of numerical algorithms. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, Springer, Heidelberg (2006)
10. Jones, N., Gomard, C., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Int. Series in Computer Science. Prentice Hall International, Englewood Cliffs (1993)
11. Martel, M.: Validation of assembler programs for DSPs: a static analyzer. In: Program analysis for software tools and engineering, pp. 8–13. ACM Press, New York (2004)
12. Martel, M.: An overview of semantics for the validation of numerical programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 59–77. Springer, Heidelberg (2005)
13. Martel, M.: Semantics of roundoff error propagation in finite precision calculations. Journal of Higher Order and Symbolic Computation 19, 7–30 (2006)
14. Sterbenz, P.H.: Floating-point Computation. Prentice-Hall, Englewood Cliffs (1974)