Boosting local consistency algorithms over oating-point numbers

Mohammed Said Belaid, Claude Michel, Michel Rueher

Accepted for publication at CP'12 (18th International Conference on Principles and Practice of Constraint Programming)

Boosting local consistency algorithms over floating-point numbers*

Mohammed Said Belaid, Claude Michel, and Michel Rueher

I3S (UNS/CNRS)
2000, route des Lucioles - Les Algorithmes - bât. Euclide B - BP 121
06903 Sophia Antipolis Cedex - France
{MSBelaid, Claude.Michel}@i3s.unice.fr, Michel.Rueher@gmail.com

Abstract. Solving constraints over floating-point numbers is a critical issue in numerous applications notably in program verification. Capabilities of filtering algorithms over the floating-point numbers ($\mathcal{F}$) have been so far limited to 2b-consistency and its derivatives. Though safe, such filtering techniques suffer from the well known pathological problems of local consistencies, e.g., inability to efficiently handle multiple occurrences of the variables. These limitations also have their origins in the strongly restricted floating-point arithmetic. To circumvent the poor properties of floating-point arithmetic, we propose in this paper a new filtering algorithm, called FPLP, which relies on various relaxations over the real numbers of the problem over $\mathcal{F}$. Safe bounds of the domains are computed with a mixed integer linear programming solver (MILP) on safe linearizations of these relaxations. Preliminary experiments on a relevant set of benchmarks are promising and show that this approach can be effective for boosting local consistency algorithms over $\mathcal{F}$.

1 Introduction

Critical systems are more and more relying on floating-point (FP) computations. For instance, embedded systems are typically controlled by software that store measurements and environment data as floating-point number ($\mathcal{F}$). The initial values and the results of all operations must therefore be rounded to some nearby float. This rounding process can lead to significant changes, and, for example, can modify the control flow of the program. Thus, the verification of programs performing FP computations is a key issue in the development of critical systems.

Methods for verifying programs performing FP computations are mainly derived from standard program verification methods. Bounded model checking (BMC) techniques have been widely used for finding bugs in hardware design [3] and software [11]. SMT solvers are now used in most of the state-of-the-art BMC tools to directly work on high level formula (see [2, 9, 11]). The bounded model checker CBMC encodes each FP operation of the program with a set of logic

* This work was partially supported by ANR VACSIM (ANR-11-INSE-0004), ANR AEOLUS (ANR-10-SEGI-0013) and OSEO ISI PAJERO projects.

functions on bit-vectors which requires thousands of additional variables and becomes quickly intractable [6]. Tools based on abstract interpretation [10, 22] can show the absence of run-time errors (e.g., division by zero) on program working with FP numbers. Tools based on abstract interpretation are safe since they over-approximate FP computations. However, over-approximations may be very large and these tools may generate many false alerts, and thus reject many valid programs. For instance, Chen's polyhedral abstract domains [7] rely on coarse approximations of floating-point operations that do not take advantage of the rounding mode. Constraint programming (CP) has also been used for program testing [13, 14] and verification [8]. CP offers many benefits like the capability to deduce information from partially instantiated problems or to exhibit counter-examples. The CP framework is very flexible and simplifies the integration of new solvers for handling a specific domain, for instance FP solvers. However, it is important to understand that solvers over real numbers ($\mathcal{R}$) cannot correctly handle FP arithmetic. Dedicated constraint solvers are required in safe CP-based framework and BMC-SMT tools for testing or verifying numerical software[1].

Techniques to solve FP constraints are based on adaptations of classical consistencies (e.g. box-consistency and 2B-consistency) over $\mathcal{R}$ [21], [20, 5]. However FP solvers based on these techniques do not really scale up to large constraint systems. That is why we introduce here a new method to handle constraints over the FP numbers by taking advantage of solvers over $\mathcal{R}$. The basic tenet is to build correct but tight relaxations over $\mathcal{R}$ of the FP operations. To ensure the tightness of the result, each FP operation is approximated according to its rounding mode. For example, assume that x and y are positive normalized FP numbers[2], then the FP product $x \otimes y$ with a rounding mode set to $-\infty$, is bounded by $\alpha \times (x \times y) < x \otimes y \leq x \times y$ where $\alpha = 1/(1 + 2^{-p+1})$ and p is the size of the significand. Approximations for special cases have also been refined, e.g., for the addition with a rounding mode set to zero, or for the multiplication by a constant.

Using these relaxations, a problem over the FP numbers is first translated into a set of nonlinear constraints over $\mathcal{R}$. A linearization of the nonlinear constraints is then applied to obtain a mixed integer linear problem (MILP) over $\mathcal{R}$. In this process, binary variables are used to handle concave domains to prevent too loose over-approximations. This last set of constraints can directly be solved by available MILP solvers over $\mathcal{R}$ which are relieved from the drawbacks of FP arithmetic. Efficient MILP solvers rely on FP computations and thus, might miss some solutions. In order to ensure a safe behavior of our algorithm, correct rounding directions are applied to the relaxation coefficients [19, 4] and a

[1] See FPSE (http://www.irisa.fr/celtique/carlier/fpse.html), a solver for FP constraints coming from C programs.

[2] A FP number is a triple (s, e, m) where s is the sign, e the exponent and m the significand. Its value is given by $(-1)^s \times 1.m \times 2^e$. r and p are the size of the exponent and the significand. The IEEE standard 754 defines the single format with $(r, p) = (8, 23)$ and the double format with $(r, p) = (11, 52)$. A normalized FP number's significand has no non-zero digits to the left of the decimal point and a non-zero digit just to the right of the decimal point.

procedure [23] to compute a safe minimizer from the unsafe result of the MILP solver is also applied. Preliminary experiments are promising and this new filtering technique should really help to scale up all verifications tools that uses a FP solver.

Our method relies on a high level representation of the FP operations and, thus, does not suffer from the same drawbacks than bit vector encoding. The bit vector encoding used in CBMC generates thousands of additional binary variables for each FP operation of the program. For example, an addition of two 32 bits floats requires 2554 binary variables [6]. The mixed approximations proposed in [6] reduce the number of additional binary variables significantly but the resulting system remains expensive in memory consumption. For instance, a single addition with only 5 bits of precision still requires 1035 additional variables. Our method does also generate additional variables: temporary variables are used to decompose complex expressions into elementary operations over the FP numbers and some binary variables are used to handle the different cases of our relaxations. However, the number of generated variables is negligible compared to the ones required by a bit vector encoding.

1.1 An illustrative example

Before going into the details, let us illustrate our approach on a very simple example. Consider the simple constraint

$$z = x \oplus y \ominus x \tag{1}$$

where x, y and z are 32 bits FP variables, and $\oplus$ and $\ominus$ are the addition and the subtraction over $\mathcal{F}$, respectively. Over the real numbers, such an expression can be simplified to $z = y$. However, this is not true with FP numbers. For example, over $\mathcal{F}$ and with a rounding mode set to the nearest, $10.0 \oplus 10.0^{-8} \ominus 10.0$ is not equal to 10.0^{-8} but to 0. This absorption phenomenon illustrates why expressions over the FP numbers cannot be simplified in the same way than expressions over the real numbers.

Now, let us assume that $x \in [0.0, 10.0]$, $y \in [0.0, 10.0]$ and $z \in [0.0, 10.0^8]$. FP2B, a 2B-consistency [16] algorithm adapted to FP constraints [20], first performs forward propagation of the domains of x and y on the domain of z using an interval arithmetic where interval bounds are computed with a rounding mode set to the nearest. Backward propagation being of no help here, the filtering process yields:

$$x \in [0.0, 10.0],\ y \in [0.0, 10.0],\ z \in [0.0, 20.0]$$

This poor filtering is due to the fact that 2B-consistency algorithms cannot handle efficiently constraints with multiple occurrences of the variables. A stronger consistency like 3B-consistency [16] will reduce the domain of z to the interval $[0.0, 10.018352508544921 88]$. However, 3B-consistency will fail to reduce the domain of z when x and y occur more than two times, like in $z = x \oplus y \ominus x \ominus y \oplus x \oplus y \ominus x$.

Algorithm FPLP, introduced in this paper, first builds safe nonlinear relaxations over $\mathcal{R}$ of the constraints over $\mathcal{F}$ derived from the program. Of course, these relaxations are computed according to the rounding mode. Applied to constraint (1), it yields the following relaxations over $\mathcal{R}$:

$$\begin{cases} (1 - \frac{2^{-p}}{(1-2^{-p})})(x+y) \leq tmp1 \\ tmp1 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(x+y) \\ (1 - \frac{2^{-p}}{(1-2^{-p})})(tmp1 - x) \leq tmp2 \\ tmp2 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(tmp1 - x) \\ z = tmp2 \end{cases}$$

where p is the size of the significand of the FP variables. $tmp1$ approximates the result of the operation $x \oplus y$ by means of two planes over $\mathcal{R}$ which encompass all the results of this addition over $\mathcal{F}$. $tmp2$ does the same for the subtraction. Some relaxations, like the one of the product, include nonlinear terms. In such a case, a linearization process is applied to get a MILP. Once the problem is fully linear, a MILP solver is used to shrink the domain of each variable, respectively, minimizing and maximizing it.

FPLP, which stands for Floating-Point Linear Program, implements the algorithm previously sketched. A call to FPLP on constraint (1) immediately yields:

$$x \in [0, 10],\ y \in [0, 10],\ z \in [0, 10.0000023841859]$$

which is a much tighter result than the one computed by FP2B. Contrary to 3B-consistency, FPLP still gives the same result with FPLP provides the same result for constraint $z = x \oplus y \ominus x \ominus y \oplus x \oplus y \ominus x$ whereas 3B-consistency cannot reduce the upper bound of z on the latter constraint.

1.2 Outline of the paper

The rest of this paper is organized as follows: the next section introduces the nonlinear relaxations over $\mathcal{R}$ of the constraints over $\mathcal{F}$. The following section shows how the nonlinear terms of the relaxations are linearized. Then, the filtering algorithm is detailed and the results of our experiments are given before concluding the paper.

2 Relaxations of FP constraints

This section introduces nonlinear relaxations over $\mathcal{R}$ of the FP constraints from the initial problem. These relaxations are the cornerstone of the filtering process described in this paper. They must be *correct*, i.e., they must preserve the whole set of solutions of the initial problem, and *tight*, i.e., they should enclose the smallest amount of non FP solutions.

These relaxations are built using two techniques: the *relative error* and the *correctly rounded* operations. The former is a technique frequently used to analyze the precision of the computation. The latter property is ensured by any

IEEE 754 compliant implementation of the FP arithmetic: a correctly rounded operation is an operation whose result over $\mathcal{F}$ is equal to the rounding of the result of the equivalent operation over $\mathcal{R}$. In other word, let x and y be two FP numbers, $\odot$ and $\cdot$, respectively, an operation over $\mathcal{F}$ and its equivalent over $\mathcal{R}$, if $\odot$ is correctly rounded then, $x \odot y = round(x \cdot y)$.

In the rest of this section, we first detail how to build these relaxations for a specific case before defining the relaxations in the general cases. Then, we will show how the different cases can be simplified.

2.1 A specific case

In order to explain how these relaxations are built, let us consider the case where an operation is computed with a rounding mode set to $-\infty$ and the result of this operation is a positive and normalized FP number. Such an operation, denoted $\odot$, could be any of the four basic binary operations from the FP arithmetic. The operands are all supposed to have the same FP type, i.e., either float, double or long double. Then, the following property holds:

Proposition 1. *Let x and y be two FP numbers whose significand is represented by p bits. Assume that the rounding mode is set to $-\infty$ and that the result of $x \odot y$ is a normalized positive FP numbers smaller than max_f, the biggest FP number, then the following property holds:*

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq (x \cdot y)$$

where $\odot$ is a basic operation over the FP numbers and, $\cdot$ is the equivalent operation over the real numbers.

Proof. Since IEEE 754 basic operations are correctly rounded and the rounding mode is set to $-\infty$, we have:

$$x \odot y \leq x \cdot y < (x \odot y)^+ \tag{2}$$

$(x \odot y)^+$, the successor of $(x \odot y)$ within the set of FP numbers, can be computed by

$$(x \odot y)^+ = (x \odot y) + ulp(x \odot y)$$

as, ulp, which stands for unit in the last place, is defined by $ulp(x) = x^+ - x$. Thus, it results from (2) that

$$x \odot y \leq x \cdot y < (x \odot y) + ulp(x \odot y)$$

From the second inequality, we have

$$\frac{1}{x \odot y + ulp(x \odot y)} < \frac{1}{x \cdot y}$$

By multiplying each side of the inequality by $x \odot y$ – which is a positive number – we get

$$\frac{x \odot y}{x \odot y + ulp(x \odot y)} < \frac{x \odot y}{x \cdot y}$$

By multiplying each side of the above inequality by -1 and by adding one to each side, we obtain

$$1 - \frac{x \odot y}{x \cdot y} < 1 - \frac{x \odot y}{x \odot y + ulp(x \odot y)} = \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)} \tag{3}$$

Now, consider ϵ, the relative error defined by

$$\epsilon = \left| \frac{real\_value - float\_value}{real\_value} \right|$$

ϵ is the absolute value of the difference between the result over $\mathcal{R}$ and the result over $\mathcal{F}$ divided by the result over $\mathcal{R}$. In the considered case, the result of $x \odot y$ being a positive normalized floating-point number and $x \cdot y \geq x \odot y$, the relative error is given by

$$0 \leq \epsilon = \frac{x \cdot y - x \odot y}{x \cdot y} = 1 - \frac{x \odot y}{x \cdot y}$$

Thus, thanks to (3), we have

$$0 \leq \epsilon < \frac{ulp(x \odot y)}{x \odot y + ulp(x \odot y)}$$

z, the result of the operation $x \odot y$, is a binary positive and normalized FP number that can be written $z = 1.m_z 2^{e_z}$, where m_z has p bits. Moreover, $ulp(z) = 2^{-p+1}2^{e_z}$. Therefore,

$$0 \leq \epsilon < \frac{2^{-p+1}2^{e_z}}{m_z 2^{e_z} + 2^{-p+1}2^{e_z}} = \frac{2^{-p+1}}{m_z + 2^{-p+1}}$$

The value of the significand of a normalized FP number belongs to the interval $[1.0, 2.0[$. An upper bound of the relative error ϵ is given by the minimum of $m_z + 2^{-p}$ which is reached when $m_z = 1$. Thus

$$0 \leq \epsilon < \frac{2^{-p+1}}{1 + 2^{-p+1}}$$

Since we have

$$\epsilon = \frac{x \cdot y - x \odot y}{x \cdot y}$$

we have

$$0 \leq \frac{x \cdot y - x \odot y}{x \cdot y} < \frac{2^{-p+1}}{1 + 2^{-p+1}}$$

and

$$0 \leq x \cdot y - x \odot y < (x \cdot y)\frac{2^{-p+1}}{1 + 2^{-p+1}}$$

Rounding mode	Negative normalized	Negative denormalized	Positive denormalized	Positive normalized
to $-\infty$	$[(1+2^{-p+1})z_r, z_r]$	$[z_r - min_f, z_r]$	$[z_r - min_f, z_r]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
to $+\infty$	$[z_r, \frac{1}{(1+2^{-p+1})}z_r]$	$[z_r, z_r + min_f]$	$[z_r, z_r + min_f]$	$[z_r, (1+2^{-p+1})z_r]$
to 0	$[z_r, \frac{1}{(1+2^{-p+1})}z_r]$	$[z_r - min_f, z_r]$	$[z_r, z_r + min_f]$	$[\frac{1}{(1+2^{-p+1})}z_r, z_r]$
to nearest	$[(1+\frac{2^{-p}}{(1+2^{-p})})z_r,$ $(1-\frac{2^{-p}}{(1-2^{-p})})z_r]$	$[z_r - \frac{min_f}{2},$ $z_r + \frac{min_f}{2}]$	$[z_r - \frac{min_f}{2}$ $z_r + \frac{min_f}{2}]$	$[(1-\frac{2^{-p}}{(1-2^{-p})})z_r,$ $(1+\frac{2^{-p}}{(1+2^{-p})})z_r]$

Table 1. Relaxations of $x \odot y$ for each rounding mode where $z_r = x \cdot y$.

By multiplying each side of the inequality by -1 and adding $x \cdot y$ to each side, we finally obtain

$$\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq x \cdot y$$

□

2.2 Generalization

Table 1 summarizes the relaxations for each rounding mode in the different cases, i.e., positive or negative FP numbers, as well as, normalized and denormalized FP numbers. Each case has a dedicated correct and tight approximation built in a way similar to the one of the case detailed in the previous subsection.

Note that tighter approximations for specific cases could also be computed. For example, the approximation of an addition with a rounding mode sets to $\pm\infty$ could be slightly improved. In a similar way, the structure of the problem is another source of improvements of the approximations. For example, $2 \otimes x$ being exactly computed[3], it can directly be evaluated over $\mathcal{R}$.

2.3 Simplified relaxations

The main issue with the previous relaxations is that the solving process will have to handle the different cases. As a result, for n basic operations, the solver has to deal with 4^n potential combinations of the relaxations. To decrease substantially this complexity, we provide here a combination of the four cases of each rounding mode into a single case.

Let us first consider the case where the rounding mode is set to $-\infty$:

Proposition 2. *Let x and y be two FP numbers whose significand size is p and, assume that the rounding mode is set to $-\infty$ and, that $-max_f < x \odot y < max_f$, then,*

$$z_r - 2^{-p+1}|z_r| - min_f \leq x \odot y \leq z_r$$

[3] Provided that no overflow occurs.

where min_f is the smallest positive FP number, $\odot$ and $\cdot$ are respectively a basic binary operation over $\mathcal{F}$ and its equivalent over $\mathcal{R}$, and $z_r = x \cdot y$.

Proof. In a first step, the normalized and denormalized approximations are combined. If $z_r > 0$ then $\frac{1}{1+2^{-p+1}} z_r < z_r$. Thus,

$$\frac{1}{1+2^{-p+1}} z_r - min_f < z_r - min_f$$

and

$$\frac{1}{1+2^{-p+1}} z_r - min_f < \frac{1}{1+2^{-p+1}} z_r$$

Therefore,

$$\frac{1}{1+2^{-p+1}} z_r - min_f < x \odot y \leq z_r,\ z_r \geq 0$$

When $z_r \leq 0$, we get

$$(1+2^{-p+1}) z_r - min_f < x \odot y \leq z_r,\ z_r \leq 0$$

These two approximations can be rewritten as follows,

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}} z_r - min_f < x \odot y \leq z_r, z_r \geq 0 \\ z_r + 2^{-p+1} z_r - min_f < x \odot y \leq z_r, \quad z_r \leq 0 \end{cases}$$

To combine the negative and positive approximations together we can use the absolute value:

$$\begin{cases} z_r - \frac{2^{-p+1}}{1+2^{-p+1}} |z_r| - min_f < x \odot y \leq z_r, z_r \geq 0 \\ z_r - 2^{-p+1} |z_r| - min_f < x \odot y \leq z_r, \quad z_r \leq 0 \end{cases}$$

As $max\{\frac{2^{-p+1}}{1+2^{-p+1}}, 2^{-p+1}\} = 2^{-p+1}$, we get

$$z_r - 2^{-p+1} |z_r| - min_f \leq x \odot y \leq z_r$$

□

The same reasoning holds for other rounding modes. Table 2 summarizes the simplified relaxations for each rounding mode. Note that these approximations define concave sets.

3 Linearization of the relaxations

The relaxations introduced in the previous section contain nonlinear terms that cannot be directly handled by a MILP solver. In this section, we describe how these terms are approximated by sets of linear constraints.

Rounding mode	The approximation of $x \odot y$
to $-\infty$	$[z_r - 2^{-p+1}\lvert z_r\rvert - min_f, z_r]$
to $+\infty$	$[z_r, z_r + 2^{-p+1}\lvert z_r\rvert + min_f]$
to 0	$[z_r - 2^{-p+1}\lvert z_r\rvert - min_f,$ $z_r + 2^{-p+1}\lvert z_r\rvert + min_f]$
to the nearest	$[z_r - \frac{2^{-p}}{(1-2^{-p})}\lvert z_r\rvert - \frac{min_f}{2},$ $z_r + \frac{2^{-p}}{(1-2^{-p})}\lvert z_r\rvert + \frac{min_f}{2}]$

Table 2. Simplified relaxations of $x \odot y$ for each rounding mode (with $z_r = x \cdot y$).

3.1 Absolute value linearization

Simplified relaxations that allow to handle all numerical FP values with a single set of two inequalities require absolute values. Absolute values can either be loosely approximated by three linear inequalities or by a tighter decomposition based on big M rewriting method:

$$\begin{cases} z = z_p - z_n \\ |z| = z_p + z_n \\ 0 \leq z_p \leq M \times b \\ 0 \leq z_n \leq M \times (1-b) \end{cases}$$

where b is a boolean variable, z_p and z_n are real positive variables and, M is a FP number such that $M \geq max\{|\underline{z}|, |\overline{z}|\}$. The method separates z_p, the positive values of z, from z_n, its negative values. When $b = 1$, z gets its positive values and we have $z = z_p = |z|$. If $b = 0$, z gets its negative values and we have $z = -z_n$ and $|z| = z_n$.

If the underlying MILP solver allows indicator constraints, the two last set of inequalities can be replaced by:

$$\begin{cases} b = 0 \rightarrow z_p = 0 \\ b = 1 \rightarrow z_n = 0 \end{cases}$$

3.2 Linearization of nonlinear operations

Bilinear terms, square terms, and quotient linearizations are based on standard techniques used by Sahinidis et al [24]. They have been also used in the Quad system [15] designed to solve constraints over the real numbers. $x \times y$ is linearized according to Mc Cormick [18]:

Let $x \in [\underline{x}, \overline{x}]$ and $y \in [\underline{y}, \overline{y}]$, then

$$\begin{cases} z - \underline{x}y - \underline{y}x + \underline{x}\underline{y} \geq 0 \\ -z + \underline{x}y + \overline{y}x - \underline{x}\overline{y} \geq 0 \\ -z + \overline{x}y + \underline{y}x - \overline{x}\underline{y} \geq 0 \\ z - \overline{x}y - \overline{y}x + \overline{x}\overline{y} \geq 0 \end{cases}$$

These linearizations have been proved to be optimal by Al-Khayyal and Falk [1].

Each time $x = y$, i.e., in case of $z = x \otimes x$, the linearization can be improved. x^2 convex hull is underestimated by all the tangents at x^2 curve between $\underline{x}$ and $\overline{x}$ and overestimated by the line that join $(\underline{x}, \underline{x}^2)$ to $(\overline{x}, \overline{x}^2)$. A good balance is obtained with the two tangents at the bounds of x. Thus, x^2 linearization yields:

$$\begin{cases} z + \underline{x}^2 - 2\underline{x}x \geq 0 \\ z + \overline{x}^2 - 2\overline{x}x \geq 0 \\ (\underline{x} + \overline{x})x - z - \underline{x}\overline{x} \geq 0 \\ z \geq 0 \end{cases}$$

The division takes advantage of the properties of real arithmetic: the essential observation is that $z = x/y$ is equivalent to $x = z \times y$. Therefore, Mc Cormick [18] linearizations can be used here. These linearizations need the bounds of z which can directly be computed by interval arithmetic:

$$\begin{aligned}[\underline{z}, \overline{z}] = [\nabla(min(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y})), \\ \Delta(max(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}))]\end{aligned}$$

where ∇ and Δ are respectively the rounding modes towards $-\infty$ and $+\infty$.

4 Filtering algorithm

The proposed filtering algorithm relies on the linearizations of the relaxations over $\mathcal{R}$ of the initial problem to attempt to shrink the domain of the variables by means of a MILP solver. Algorithm 1 details the steps of this filtering process.

First, function **Approximate** relaxes initial FP constraints to nonlinear constraints over $\mathcal{R}$. Then, function **Linearize** linearizes the nonlinear terms of these relaxations to get a MILP.

The filtering loop starts with a call to FP2B, a filtering process relying on an adaptation of 2B-consistency to FP constraints that attempts to reduce the bounds of the variables. FP2B propagates bound values to intermediate variables. The cost of this filtering process is quite light: it stops as soon as domain size reduction between two iterations is less than 10%. Thanks to function **UpdateLinearizations**, newly computed bounds are used to tighten the MILP. Note that this function updates variable domains as well as linearization coefficients.

After that, MILP is used to compute a lower bound and an upper bound of the domain of each variable by means of function **safeMin**. This function computes a safe global minimizer of the MILP.

This process is repeated until the percentage of reduction of the domains of the variables is lower than a given ϵ.

Algorithm 1 FPLP

1: **Function** $FPLP(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$
2: % $\mathcal{V}$: FP variables
3: % $\mathcal{D}$: Domains of the variables
4: % $\mathcal{C}$: Constraints over FP numbers
5: % ϵ: Minimal reduction between two iterations
6: $\mathcal{C}' \leftarrow$ **Approximate** $(\mathcal{C})$;
7: $\mathcal{C}'' \leftarrow$ **Linearize** $(\mathcal{C}', \mathcal{D})$;
8: $boxSize \leftarrow \sum_{x \in \mathcal{V}} (\overline{x}_{\mathcal{D}} - \underline{x}_{\mathcal{D}})$;
9: **repeat**
10: $\quad \mathcal{D}' \leftarrow$ **FP2B**$(\mathcal{V}, \mathcal{D}, \mathcal{C}, \epsilon)$;
11: $\quad$ **if** $\emptyset \in \mathcal{D}'$ **then**
12: $\quad\quad$ **return** $\emptyset$;
13: $\quad$ **end if**
14: $\quad \mathcal{C}'' \leftarrow$ **UpdateLinearizations**$(\mathcal{C}'', \mathcal{D}')$;
15: $\quad$ **for all** $x \in \mathcal{V}$ **do**
16: $\quad\quad [\underline{x}_{\mathcal{D}'}, \overline{x}_{\mathcal{D}'}] \leftarrow [$**safeMin**$(x, \mathcal{C}''),\ -$**safeMin**$(-x, \mathcal{C}'')]$;
17: $\quad\quad$ **if** $[\underline{x}_{\mathcal{D}'}, \overline{x}_{\mathcal{D}'}] = \emptyset$ **then**
18: $\quad\quad\quad$ **return** $\emptyset$;
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad oldBoxSize \leftarrow boxSize$;
22: $\quad boxSize \leftarrow \sum_{x \in \mathcal{V}} (\overline{x}_{\mathcal{D}'} - \underline{x}_{\mathcal{D}'})$;
23: $\quad \mathcal{D} \leftarrow \mathcal{D}'$
24: **until** $boxSize \geq oldBoxSize * (1 - \epsilon)$;
25: **return** $\mathcal{D}$;

4.1 Getting a safe minimizer

Using an efficient MILP solver like CPLEX to filter the domains of the variables raises two important issues related to FP computations.

First, linearization coefficients are computed with FP arithmetic and are subject to rounding errors. Therefore, to avoid the loss of solutions, special attention must be paid to the rounding directions. Correct linearizations rely on FP computations done using the right rounding directions. For instance, consider the linearization of x^2 where $\underline{x} \geq 0$ and $\overline{x} \geq 0$:

$$\begin{cases} y + \Delta(\underline{x}^2) - \Delta(2\underline{x})x \geq 0 \\ y + \Delta(\overline{x}^2) - \Delta(2\overline{x})x \geq 0 \\ \Delta(\underline{x} + \overline{x})x - y - \nabla(\underline{x}\overline{x}) \geq 0 \\ y \geq 0 \end{cases}$$

This process that ensures that all the linearizations are safe is called within the **Linearize** and **UpdateLinearizations** functions. For more details on how to compute safe coefficients see [19, 4].

				2B	3B		FPLP (without 2B)		FPLP	
Program	n	n_T	n_B	$t(ms)$	$t(ms)$	%(2B)	$t(ms)$	%(2B)	$t(ms)$	%(2B)
`Absorb1`	2	1	1	TO	TO	-	3	98.91	5	98.91
`Absorb2`	2	1	1	1	24	0.00	3	100.00	4	100.00
`Fluctuat1`	3	12	2	4	156	99.00	264	99.00	172	99.00
`Fluctuat2`	3	10	2	1	4	0.00	29	0.00	21	0.00
`MeanValue`	4	28	6	3	82	97.45	530	97.46	78	97.46
`Cosine`	5	33	7	5	153	33.60	104	33.61	43	33.61
`SqrtV1`	11	140	29	9	27198	99.63	1924	100.00	1187	100.00
`SqrtV2`	21	80	17	7	TO	-	2337	100.00	1321	100.00
`SqrtV3`	5	46	8	5	573	53.80	185	54.83	82	54.83
`Sine taylor`	6	44	9	5	452	63.29	313	63.29	227	63.29
`Sine iter`	16	109	21	8	4503	39.20	5885	39.31	165	39.31
`Qurt`	6	21	3	4	26	43.56	163	43.56	38	43.56
`Poly`	6	51	9	5	1569	49.17	765	76.66	309	76.66
`Newton`	7	69	14	5	1542	45.16	479	45.16	195	45.16

Table 3. Experiments

Second, efficient MILP solvers use FP arithmetic. Thus, the computed minimizer might be wrong. The unsafe MILP solver is made safe thanks to the correction procedure introduced in [23]. It consists in computing a safe lower bound of the global minimizer. The **safeMin** function implements these corrections and return a safe minimizer of the MILP.

5 Experiments

This section compares the results of different filtering techniques for FP constraints with the method introduced in this paper. Experiments have been done on a laptop with an Intel Duo Core at 2.8Ghz and 4Gb of memory running under Linux.

Our experiments are based on the following set of benchmarks:

- `Absorb 1` detects if, in a simple addition, x absorbs y while `Absorb 2` checks if y absorbs x.
- `Fluctuat1` and `Fluctuat2` are program pathes that come from a presentation of the Fluctuat tool in [12].
- `MeanValue` returns true if an interval contains a mean value and false otherwise.
- `Cosine` is a program that computes the function $cos()$ with a Taylor formula.
- `SqrtV1` computes sqrt in $[0.5, 2.5]$ using a two variable iterative method.
- `SqrtV2` computes sqrt with a Taylor formula.

- `SqrtV3` computes the square root of $(x+1)$ using a Taylor formula. This program comes from CDFPL benchmarks[4].
- `Sine taylor` computes the function sine using a Taylor formula.
- `Sine iter` computes the function sine with an iterative method and comes from the SNU real time library[5].
- `Qurt` computes the real and imaginary roots of a quadratic equation and also comes from the SNU library.
- `Poly` tries to compare two different writings of a polynomial. This program is available on Eric Goubault web page[6]
- `Newton` computes one or two iterations of a Newton on the polynomial $x - x^3/6 + x^5/120x^7/5040$ and comes from CDFPL benchmarks.

Table 3 summarizes experiment results for the following filtering methods: FP2B, an adaptation of 2B-consistency to FP constraints that takes advantage of the property described in [17] to avoid some slow convergences, FP3B, an adaptation of 3B-consistency to FP constraints, FPLP(without FP2B), an implementation of algorithm 1 without the call to FP2B and, FPLP, an implementation of algorithm 1. First column of table 3 gives program's names, column 2 gives the number of variables of the initial problem and column 3 gives the amount of temporary variables used to decompose complex expressions in elementary operations. Column 4 gives the number of binary variables used by FPLP. For each filtering algorithm, table 3 gives the amount of milliseconds required to filter the constraints (columns $t(ms)$). For all filtering algorithm but FP2B, table 3 gives also the percentage of reduction compared to the reduction obtained by FP2B (columns %(FP2B)). The time out (TO) was set to 2 minutes.

The results from table 3 show that FPLP achieves much better domain reductions than 2B-consistency and 3B-consistency filtering algorithms. FPLP requires more times than FP2B but the latter achieves a very weaker pruning on theses benchmarks. This is exemplified by the two `Absorb1` and `SqrtV1` benches. Here, FP2B suffers from the multiple occurrences of the variables. FPLP also consistently outperforms FP3B : it almost always provides much smaller domains and it requires much less time.

A comparison of FPLP with and without a call to FP2B shows that a cooperation between these two filtering methods can significantly decrease the computation time but does not change the filtering capabilities.

6 Conclusion

In this paper, we have introduced a new filtering algorithm for handling constraints over FP numbers. This algorithm benefits from the linearizations of the relaxations over $\mathcal{R}$ of the initial constraints over $\mathcal{F}$ to reduce the domains of the variables with a MILP solver. Experiments show that FPLP drastically improves

[4] See http://www.cprover.org/cdfpl/.
[5] See http://archi.snu.ac.kr/realtime/
[6] See http://www.lix.polytechnique.fr/~goubault/.

the filtering process, especially when combined with a FP2B filtering process. MILP benefits from a more global view of the constraint system than local consistencies, and thus provides an effective way to handle multiple occurrences of variables.

Additional experiments are required to better understand the interactions between the two algorithms and to improve their performances.

References

1. F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, pages 8:2:273–286, 1983.
2. Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using smt solvers instead of sat solvers. *Int. J. Softw. Tools Technol. Transf.*, 11:69–83, January 2009.
3. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, 1999. Springer-Verlag.
4. Glencora Borradaile and Pascal Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 2005.
5. Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Softw. Test., Verif. Reliab.*, 16(2):97–121, 2006.
6. Angelo Brillout, Daniel Kroening, and Thomas Wahl. Mixed abstractions for floating-point arithmetic. In *Proceedings of FMCAD 2009*, pages 69–76. IEEE, 2009.
7. Liqian Chen, Antoine Miné, and Patrick Cousot. A sound floating-point polyhedra abstract domain. In *Proceedings of the 6th Asian Symposium on Programming Languages and Systems*, APLAS '08, pages 3–18, Berlin, Heidelberg, 2008. Springer-Verlag.
8. Hélène Collavizza, Michel Rueher, and Pascal Hentenryck. CPBPV: a constraint-programming framework for bounded program verification. *Constraints*, 15(2):238–264, 2010.
9. Lucas Cordeiro, Bernd Fischer, and Joao Marques-Silva. Smt-based bounded model checking for embedded ansi-c software. *IEEE Transactions on Software Engineering*, May 2011.
10. Patrick Cousot, Radhia Cousot, Jerome Feret, Antoine Miné, Laurent Mauborgne, David Monniaux, and Xavier Rival. Varieties of static analyzers: A comparison with astree. In *TASE '07*, pages 3–20. IEEE, 2007.
11. Malay K Ganai and Aarti Gupta. Accelerating high-level bounded model checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 794–801, New York, NY, USA, 2006. ACM.
12. K. Ghorbal, E. Goubault, and S. Putot. A logical product approach to zonotope intersection. In *Computer Aided Verification*, pages 212–226. Springer, 2010.
13. Arnaud Gotlieb, Bernard Botella, and Michel Rueher. Automatic test data generation using constraint solving techniques. In *ISSTA*, pages 53–62, 1998.
14. Arnaud Gotlieb, Bernard Botella, and Michel Rueher. A clp framework for computing structural test data. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2000.

15. Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean-Pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM J. Numer. Anal*, 42:2076–2097, 2005.
16. Olivier Lhomme. Consistency techniques for numeric csps. In *IJCAI*, pages 232–238, 1993.
17. Bruno Marre and Claude Michel. Improving the floating point addition and subtraction constraints. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 360–367. Springer, 2010.
18. G.P. McCormick. Computability of global solutions to factorable nonconvex programs – part i – convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.
19. C. Michel, Y. Lebbah, and M. Rueher. Safe embedding of the simplex algorithm in a CSP framework. In *Proc. of CPAIOR 2003, CRT, Université de Montréal*, pages 210–220, 2003.
20. Claude Michel. Exact projection functions for floating point number constraints. In *AMAI*, 2002.
21. Claude Michel, Michel Rueher, and Yahia Lebbah. Solving constraints over floating-point numbers. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2001.
22. Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Palaiseau, France, December 2004.
23. A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A.*, page 99:283 296, 2004.
24. Hong S. Ryoo and Nikolaos V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, pages 107–138, 1996.