Filtering by ULP Maximum

Matthieu Carlier, Arnaud Gotlieb

Filtering by ULP maximum

Matthieu Carlier
INRIA Rennes Bretagne Atlantique
Rennes, France
Email: Matthieu.Carlier@inria.fr

Arnaud Gotlieb
Certus Software V&V Center
Simula Research Laboratory,
Norway, Oslo, Norway

***Abstract*—Constraint solving over floating-point numbers is an emerging topic that found interesting applications in software analysis and testing. Even for IEEE-754 compliant programs, correct reasoning over floating-point computations is challenging and requires dedicated constraint solving approaches to be developed. Recent advances indicate that numerical properties of floating-point numbers can be used to efficiently prune the search space. In this paper, we reformulate the Marre and Michel property over floating-point addition/subtraction constraint to ease its implementation in real-world floating-point constraint solvers. We also generalize the property to the case of multiplication/division in order to benefit from its improvements in more cases.**

I. Introduction

Numerical programs used in critical software systems must be thoroughly tested before being embedded within a final product. In the presence of floating-point computations, this involves 1) producing test inputs [18] targeted to reveal faults and 2) predicting the program results expected on those computations. For this latter problem, called the *oracle problem* [21], several techniques have been proposed, such as using multiple program executions to check the results in the data diversity approach [1], or using the Abstract Interpretation framework [6] to estimate the deviance of the floating-point results w.r.t. an interpretation over the reals [9], or else using perturbation techniques to evaluate the stability of a numerical program [20]. Constraint Programming techniques have also been used to tackle the *oracle problem*: [12] proposed a Gecode model to evaluate the expected results of a trading system's continuous double auction, [8] proposed using the SICStus Prolog clpfd library to generate test inputs that violate properties over imperative programs, etc. However, in the presence of floating-point computations, very few approaches addressed the former problem, namely the *test inputs generation problem*. Thirty years ago, Miller and Spooner [18] proposed to automatically find a floating-point test input that exercises a given program path, by minimizing a cost function which evaluates the distance between the currently executed path and the expected one. Their work opened the door for search-based test data generation methods [11], [15] that share great similarities with local search techniques in Constraint Programming [2]. However, these approaches are only based on program executions and do not rely on symbolic reasoning. Thus, they cannot be used to study path feasibility, i.e. to decide whether a possible execution path is feasible or not in the program. In addition, these techniques can be stuck in local minima without being able to provide a meaningful result [2]. An approach to tackle these problems combines both the program execution and the symbolic reasoning [7], but this kind of reasoning over floating-point computations is hard and requires dedicated filtering algorithms [16], [17]. In summary, solving constraints over floating-point numbers allows us to generate test inputs that exercise a selected behaviour of the program under test. This approach is currently implemented in three solvers: the FPCS solver [3], FPSE for C programs [4] which is available for experiments[1] and Gatel, the test data generator for Lustre programs [3], [13].

A promising approach to improve the filtering capabilities of constraints over floating-point variables consists in using numerical properties of floating-point computations. For linear constraints, this led to a relaxation technique where floating-point numbers and constraints are converted into constraints over the reals by using Linear Programming approaches [19]. For interval-based consistency approaches, Marre and Michel proposed in [14] to exploit the floating-point representation in filtering algorithms for both the addition and subtraction constraints.

This paper is concerned with the reformulation of the Marre and Michel property in terms of filtering by maximum ULP (Units in the Last Place), in order to ease its implementation in real-world constraint solvers such as FPSE or FPCS. In addition, we generalize the property to the multiplication and division operators for benefiting of its improvements in more cases. This generalization has an interesting impact on non-linear problems where no other techniques yet exist to improve the search space filtering.

The rest of the paper is organized as follows. Next section briefly presents the IEEE-754 standard of binary floating-point numbers and introduces the notations used throughout the paper. Section 3 recalls the basic principles of interval-based consistency techniques over floating-point variables and constraints. Section 4 presents our generalization of the

[1] `http://www.irisa.fr/celtique/carlier/fpse.html`

Marre and Michel property, while section 5 concludes the paper.

II. Preliminaries

A. IEEE-754

This section introduces the arithmetical model specified by the IEEE-754 standard for binary floating-point arithmetic [10].

IEEE-754 specifies two basic binary floating-point formats (single and double) and two extended formats. A floating-point number is noted $(-1)^s a.m \times 2^e$ where s is the sign bit, a is the hidden bit, m is the significand and e is the biased exponent. The single format occupies 32 bits (1 bit for the sign, 8 for the exponent and 23 for the significand) while the double occupies 64 bits (1 bit for the sign, 11 for the exponent and 52 for the significand). Each format defines several classes of numbers: normalized numbers, denormalized numbers, signed zeros, infinities and NaNs (which stands for Not-a-Number). For the single format, normalized numbers corresponds to an exponent value $0 < e < 255$ and a value given by the formula: $(-1)^s\ 1.m\ 2^{e-127}$. Denormalized numbers correspond to an exponent $e = 0$ and a value given by $(-1)^s\ 0.m\ 2^{-126}$ where $m \neq 0$. Note that the significand possesses a hidden bit which is 1 for normalized numbers and 0 for denormalized. Note also that the bias is equal to 127 for the single format and the exponent is -126 for denormalized numbers. There are two infinities (noted $+\mathit{INF}$, $-\mathit{INF}$ with $e = 255, m = 0$) and two signed zeros (noted $+0.0$, -0.0 with $e = 0, m = 0$). NaNs ($e = 255, m \neq 0$) are used to represent the results of invalid computations such as a division or a subtraction of two infinities. They allow the program execution to continue without being halted by an exception. IEEE-754 indicates five types of rounding directions: toward negative infinity (down), toward positive infinity (up), toward zero (chop) and toward the nearest representable value, with two flavors, tail-to-even or tail-to-away in which respectively values with even mantissa are preferred or values away from zero. The to-the-nearest tail-to-even value of a real x will be noted $\circ(x)$. All rounding modes are monotonic, i.e., $\forall x\ y \in \mathcal{R}, x \leq y \Rightarrow \circ(x) \leq \circ(y)$. The most important requirement of IEEE-754 arithmetic is the accuracy of floating-point computations: each of the following operations, add, subtract, multiply, divide, square root, remainder, conversions and comparisons, must deliver to its destination the exact result if possible or the floating-point number that requires the least modification of the exact result w.r.t. the prescribed rounding mode and the result format destination. It is said that these operations are correctly rounded For example, the single-format result of $999999995904 + 10000$ is[2] 999999995904 which is the single-format floating-point number nearest to the exact result over the reals. This example shows that the accuracy requirement of IEEE-754 does not prevent surprising results from arising (the second operand is absorbed by the addition operator).

[2]These numbers can be exactly represented by single binary FP numbers.

B. Notations

$\mathcal{R}$ denotes the set of reals while $\mathcal{F}$ denotes an idealized finite set of numerical binary floating-point numbers, defined from a given IEEE-754 format. Throughout the paper, we will consider only floats having a numerical binary representation in the single or the double format, excluding denormalized numbers and NaNs, but including $-\mathit{INF}, +\mathit{INF}$ and zeros. Considering this idealized set is advantageous to simplify the properties and avoid too technical details about denormalized numbers. But, extensions are mentioned when available. A real decimal constant (such as $1.0e12$) denotes a floating-point value, and thus, has to be understood as its nearest floating-point number (*i.e.,* as 999999995904). Henceforth x^+ (resp. x^-) denotes the smallest (resp. greatest) floating-point number strictly greater (resp. smaller) than x w.r.t. the considered IEEE-754 format. We denote *min* the exponent of the smallest normalized numbers in absolute values. Thus, the smallest positive normalized number is $1.0\ldots0 \times 2^{\mathit{min}}$. In our idealized set of numbers, *min* is the smallest possible exponent for a float. On the opposite, *max* denotes the greatest possible exponent for a float distinct from $-\mathit{INF}$ or $+\mathit{INF}$. f_{max} is the greatest representable numerical float. Its value is $1.1\ldots1 \times 2^{\mathit{max}}$ and we have $f^+_{max} = +\mathit{INF}$.

Arithmetical operations over the floats will be noted using the four operators: $\oplus$, $\ominus$, $\otimes$ and $\oslash$, corresponding respectively to $+, -, *, /$ over the reals. According to IEEE-754, they are defined with the rounding operator $\circ$ as follows:

$$x \oplus y = \circ(x + y), x \ominus y = \circ(x - y),$$
$$x \otimes y = \circ(x * y), x \oslash y = \circ(x/y)$$

$\odot$ denotes any of $\oplus$, $\ominus$, $\otimes$ and $\oslash$. A floating-point variable x will be associated an interval of possible floating-point values, noted $\mathsf{x} \in [\underline{\mathsf{x}}, \overline{\mathsf{x}}]$ where $\underline{\mathsf{x}}$ denotes the smallest float of x and $\overline{\mathsf{x}}$ its greatest value and $\underline{\mathsf{x}} \leq \overline{\mathsf{x}}$. Finally, $\mathrm{mid}(a, b)$ denotes the floating-point number at the middle of a and b, which may be a floating-point number of a wider format than of its operands.

III. Background on constraint solving over floating-point variables

A. Interval-based consistency on arithmetical constraints

[4], [16] contain formulas for projectors in a interval-based consistency approach to constraint solving over the floats. It is worth distinguishing direct from indirect projectors as constraints come from program analysis. Roughly speaking, when analyzing an imperative program, assignments are considered as if they were equality constraints, where the assigned variable is labelled with a fresh name. For example, the assignment `i++;` is translated into the

$Addition : \mathsf{z} = \mathsf{x} \oplus \mathsf{y}$	
$\overline{\mathsf{z}} = \overline{\mathsf{x}} \oplus \overline{\mathsf{y}}, \quad \underline{\mathsf{z}} = \underline{\mathsf{x}} \oplus \underline{\mathsf{y}}$	(direct)
$\overline{\mathsf{x}} = \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+) \ominus \underline{\mathsf{y}}$	(1^{st} indirect)
$\underline{\mathsf{x}} = \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-) \ominus \overline{\mathsf{y}}$	
$\overline{\mathsf{y}} = \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+) \ominus \underline{\mathsf{x}}$	(2^{nd} indirect)
$\underline{\mathsf{y}} = \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-) \ominus \overline{\mathsf{x}}$	

$Substraction : \mathsf{z} = \mathsf{x} \ominus \mathsf{y}$	
$\overline{\mathsf{z}} = \overline{\mathsf{x}} \ominus \underline{\mathsf{y}}, \quad \underline{\mathsf{z}} = \underline{\mathsf{x}} \ominus \overline{\mathsf{y}}$	(direct)
$\overline{\mathsf{x}} = \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+) \oplus \overline{\mathsf{y}}$	(1^{st} indirect)
$\underline{\mathsf{x}} = \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-) \oplus \underline{\mathsf{y}}$	
$\overline{\mathsf{y}} = \overline{\mathsf{x}} \ominus \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-)$	(2^{nd} indirect)
$\underline{\mathsf{y}} = \underline{\mathsf{x}} \ominus \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+)$	

Figure 1. Formulas for direct/indirect projectors

equality constraint $i_2 = i_1 + 1$. Hence, there are two distinct kinds of projection. The projection over variable i_2 is called *direct projection* while the projector over i_1 is called *indirect projection*. When solving constraints over floating-point variables, the formulas for direct and indirect projections may be different in order to improve the filtering results. Fig.1 recalls the formulas used for implementing the interval-based addition/subtraction projectors, while (non-optimal) formulas for product/division can be found in [4], [16].

B. The Marre and Michel property

This section presents the Marre and Michel property published in [14] for improving the filtering of the addition/subtraction projectors. The idea behind this property comes from the representation of floating-point numbers among the reals: the greater a float is, the greater the distance between it and its immediate successor is. More precisely, for a given float x with exponent n, if $\Delta = x^+ - x$, then for y of exponent $n+1$ we have $y^+ - y = 2 * \Delta$.

Fig.2 gives an intuitive view of the property for subtraction. Let $\mathsf{z} = \mathsf{y} \ominus \mathsf{x}$ and suppose that $\mathsf{z} \in [v_\mathsf{z}, v_\mathsf{z}]$ is a strictly positive constant, then the Marre and Michel property says that it exists two greatest values y_m and x_m such that $y_m \ominus x_m = v_\mathsf{z}$. It is worth noticing that y_m and x_m depends neither on y nor on x and do not necessarily belong to them. For example, the existence of x_m (similar for y_m) can be explained on Fig.2 as follows. Consider x_m^+, then for all floating-point v, $v \leq y_m \implies v - x_m^+ \leq A$ and $y_m < v \implies B < v - x_m^+$ with $A < v_\mathsf{z} < B$. Thus, x_m^+ cannot be part of the solution. Considering values greater than x_m^+ leads to equivalent inequalities with some A', B' such that $A' \leq A$ and $B \leq B'$.

property 3.1: [14] Let a variable $\mathsf{z} \in [\underline{\mathsf{z}}, \overline{\mathsf{z}}]$ such that $0 < \underline{\mathsf{z}} < \overline{\mathsf{z}}$, if $\mathsf{z} = \mathsf{x} \ominus \mathsf{y}$, then upper bounds of x and y can be computed using the following formulas. Let ζ be a normalized floating-point number of z such that

$$\zeta = \begin{cases} 1.0\ldots0 \times 2^{e_{\overline{\mathsf{z}}}} & \text{iff } e_{\underline{\mathsf{z}}} \neq e_{\overline{\mathsf{z}}} \\ \underline{\mathsf{z}} & \text{iff } \underline{\mathsf{z}} = 1.0\ldots0 \times 2^{e_{\underline{\mathsf{z}}}} \\ 1.b_2\ldots b_{i+1}0\ldots \times 2^{e_{\overline{\mathsf{z}}}} & \text{iff } \begin{array}{l} \underline{\mathsf{z}} = b_1.b_2\ldots b_i 0 b_{i+2}\ldots b_p \times 2^n \\ \overline{\mathsf{z}} = b_1.b_2\ldots b_i 1 b'_{i+2}\ldots b'_p \times 2^n \end{array} \\ & \text{with } b_{i+2} \neq b'_{i+2} \end{cases}$$

and let nb_z be the number of zeros in the significand of ζ. There do not exist any value x', strictly greater than β and y', strictly greater than α such that $x' \ominus y' = \zeta$ where

$$\alpha = 1.1\ldots1 \times 2^{e_\zeta + nb_z}$$
$$\beta = \alpha \oplus \zeta$$

The above theorem can be extended to the case where ζ is a denormalized floating-point number by considering that the hidden bit is 0 and by shifting the significand to its first non-zero bit. In the case where z contains only strictly non-negative numbers, the above theorem can be used as is for improving the filtering algorithm of both the addition/subtraction constraints. When z contains only strictly negative numbers, an argument related to the symmetry of floating-point representation can be used as well. For similar reasons, the Marre and Michel property can be used for the filtering algorithm of both lower bounds of variables x and y. However, the above theorem cannot be extended to an interval of z that contains zero. Fig.3 summarizes the formulas obtained with the Marre and Michel property over floating-point addition/subtraction constraints after exploiting symmetries.

IV. Filtering by maximum ULP

This section reformulates the Marre and Michel property by considering the properties that should be verified by a function $\delta_\odot$ to deduce an optimal interval-consistency based filtering algorithm for $\odot$. It also generalizes the property to product/division projectors. In this paper, the filtering algorithms that result from this generalization are collectively called *filtering by maximum ULP* to refer to the maximal distance between two successive floats.

A. Upper bound

Let $\overline{\delta}_\odot : \mathcal{F} \to \mathcal{F}^+$ be a function that satisfies the following properties:

$$\forall z \in \mathcal{F}, \quad z \neq 0 \Rightarrow \exists y, \overline{\delta}_\odot(z) \odot y = z$$
$$\forall z\ z' \in \mathcal{F},\ z \neq 0 \Rightarrow z' > \overline{\delta}_\odot(z) \Rightarrow \nexists y, z' \odot y = z$$

Roughly speaking, those properties say that $\overline{\delta}_\odot$ returns the smallest float that permits one to recover z. The following is entailed by both properties:

property 4.1: Let $\{v_1, \ldots, v_n\}$ be a set of non-zero floats and m a float such that $\forall i, \overline{\delta}_\odot(m) \geq \overline{\delta}_\odot(v_i)$. Then, for all floats $m' > \overline{\delta}_\odot(m)$, there does not exist a float y such that $m' \odot y = v_i$.

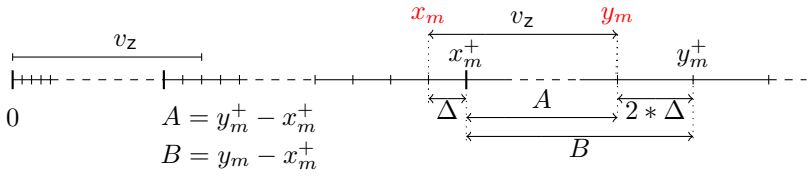

Figure 2. Existence of a limit such that $x^+ - x \geq v_z$

	$\mathsf{z} > 0$	$\mathsf{z} < 0$
$\mathsf{z} = \mathsf{x} \ominus \mathsf{y}$	$\mathsf{x} \in [-\alpha, \beta], \mathsf{y} \in [-\beta, \alpha]$	$\mathsf{x} \in [-\beta, \alpha], \mathsf{y} \in [-\alpha, \beta]$
$\mathsf{z} = \mathsf{x} \oplus \mathsf{y}$	$\mathsf{x} \in [-\alpha, \beta], \mathsf{y} \in [-\alpha, \beta]$	$\mathsf{x} \in [-\alpha, \beta], \mathsf{y} \in [-\alpha, \beta]$

Figure 3. Summary of the filtering formulas for the Marre and Michel property

As a consequence, by considering the constraint $\mathsf{z} = \mathsf{x} \odot \mathsf{y}$ where $0 \notin \mathsf{z}$, we got that the smallest float m greater than the upper bound of the co-domain of $\overline{\delta}_\odot$ on z (i.e., $\forall v_\mathsf{z} \in \mathsf{z}, \overline{\delta}_\odot(m) \geq \overline{\delta}_\odot(v_\mathsf{z})$) such that $\overline{\delta}_\odot(m)$ is the upper bound of x. It is worth says that m belongs to z.

Lastly, if we have a function $\overline{\delta'_\odot}$ which satisfies all the above mentioned properties but where the operands of $\odot$ are inverted in the conclusions, i.e.,

$$\begin{array}{ll} \forall z \in \mathcal{F}, & z \neq 0 \Rightarrow \quad \exists x, x \odot \overline{\delta'_\odot}(z) = z \\ \forall z\ z' \in \mathcal{F}, & z \neq 0 \Rightarrow z' > \overline{\delta'_\odot}(z) \Rightarrow \nexists x, x \odot z' = z \end{array}$$

then $\overline{\delta'_\odot}$ permits one to estimate the upper bound of y in the above constraint. Moreover, when $\odot$ is a commutative operator (e.g., $\oplus$, $\otimes$), the properties of $\overline{\delta}_\odot$ and $\overline{\delta'_\odot}$ are equivalent and $\overline{\delta}_\odot = \overline{\delta'_\odot}$.

B. Lower bound

For computing the lower bound, we consider the function $\underline{\delta}_\odot : \mathcal{F} \to \mathcal{F}^-$ having the following properties:

$$\begin{array}{ll} \forall z \in \mathcal{F}, & z \neq 0 \Rightarrow \quad \exists y, \underline{\delta}_\odot(z) \odot y = z \\ \forall z\ z' \in \mathcal{F}, & z \neq 0 \Rightarrow z' < \underline{\delta}_\odot(z) \Rightarrow \nexists y, z' \odot y = z \end{array}$$

As for upper bound, these two properties entailed a property similar to property 4.1. Hence, considering the constraint $\mathsf{z} = \mathsf{x} \odot \mathsf{y}$, the value of z which minimizes $\underline{\delta}_\odot$ permits to deduce the lower bound of x.

example 4.1: Consider the following constraints,

$\mathsf{x} \in [-1.0 \times 2^{50}, 1.0 \times 2^{50}]$ $\quad \mathsf{y} \in [-1.0 \times 2^{30}, 1.0 \times 2^{30}]$

$\mathsf{z} \in [1.0, 2.0]$ $\quad \mathsf{z} = \mathsf{x} \oplus \mathsf{y}$

we obtain the following results after filtering

without *maximum ULP*
$\overline{\mathsf{x}} = \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+) \ominus \underline{\mathsf{y}} = 1.0 \times 2^{30}$
$\underline{\mathsf{x}} = \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-) \ominus \overline{\mathsf{y}} = -1.0 \times 2^{30}$
$\overline{\mathsf{y}} = \text{mid}(\overline{\mathsf{z}}, \overline{\mathsf{z}}^+) \ominus \underline{\mathsf{x}} = 1.0 \times 2^{50}$
$\underline{\mathsf{y}} = \text{mid}(\underline{\mathsf{z}}, \underline{\mathsf{z}}^-) \ominus \overline{\mathsf{x}} = -1.0 \times 2^{50}$

using *maximum ULP*
$\zeta = 1.0\ldots0 \times 2^1$ and then $nb_z = 23$
$\alpha = 1.1\ldots1 \times 2^{1+nb_z}$
$\beta = \alpha \oplus \zeta = 1.0\ldots0 \times 2^{25}$
$\overline{\mathsf{x}} = \beta = 1.0\ldots0 \times 2^{25}$
$\underline{\mathsf{x}} = -\alpha = -1.1\ldots1 \times 2^{24}$
$\overline{\mathsf{y}} = \beta = 1.0\ldots0 \times 2^{25}$
$\underline{\mathsf{y}} = -\alpha = -1.1\ldots1 \times 2^{24}$

This example shows that the Marre and Michel property (called *filtering by maximum ULP*, see below) permits one to get tighter results than classical interval-consistency based filtering. Note that other trivial examples show that the opposite may also arise and therefore it is worth computing the intersection of both filtering to get optimal results. Other examples that show those phenomena can be found in [5].

C. Filtering by maximum ULP on addition/subtraction

This section introduces only the functions $\underline{\delta}_\oplus$ and $\overline{\delta}_\oplus$. The functions $\underline{\delta}_\ominus$ and $\overline{\delta}_\ominus$ can be deduced from symmetries, as explained in Sec. III-B and [14].

definition 4.1: The function $\overline{\delta}_\oplus$ is defined as follows (i and n are defined such $|z| = 1.z_1 \ldots z_i 10\ldots0 \times 2^n$):

$$\begin{array}{ll} \overline{\delta}_\oplus(0) = 0 & \\ \overline{\delta}_\oplus(z) = z + 1.1\ldots1 \times 2^{n+(p-i-1)} & \text{for } z > 0 \\ \overline{\delta}_\oplus(z) = 1.1\ldots1 \times 2^{n+(p-i-1)} & \text{for } z < 0 \end{array}$$

which is always a float as $z + 1.1\ldots1 \times 2^{n+(p-i-1)} =$

$$\begin{array}{rcl} 0.0\ldots01z_1\ldots z_i & 10\ldots0 & \times 2^{n+(p-i)} \\ \oplus\ 0.1\ldots11\ 1\ldots1 & 1 & \times 2^{n+(p-i)} \\ \hline 1.0\ldots01z_1\ldots z_i & 00\ldots0 & \times 2^{n+(p-i)} \end{array}$$

$\overline{\delta}_\oplus$ respects both properties stating for all non-zero z, $\overline{\delta}_\oplus(z)$ returns the smallest float that permits to recover z. $\underline{\delta}_\oplus$ is defined as $\underline{\delta}_\oplus(z) = -\overline{\delta}_\oplus(-z)$ and its properties are entailed by properties of $\overline{\delta}_\oplus$.

definition 4.2: For a variable z of normalized floating-point values such that 0 does not belong to z domain, the value in z maximizing $\overline{\delta}_\oplus$ (and minimizing $\underline{\delta}_\oplus$) is defined by destr($\mathsf{z}$) as:

$$\begin{aligned}
\text{destr}(\mathsf{z}) &= 1.0\ldots0\times 2^n \qquad \text{if } e_{\underline{\mathsf{z}}} \neq e_{\overline{\mathsf{z}}} \text{ with } n = e_{\overline{\mathsf{z}}}\\
\text{destr}(\mathsf{z}) &= 1.b_2\ldots b_i a 0\ldots\times 2^n\\
&\quad \text{if } \underline{\mathsf{z}} = 1.b_2\ldots b_i b_{i+1}\ldots\times 2^n \text{ with } b_{i+1}\neq b'_{i+1}\\
&\quad \overline{\mathsf{z}} = 1.b_2\ldots b_i b'_{i+1}\ldots\times 2^n\\
&\quad \text{and } a = \begin{cases} 0 & \text{if } 1.b_2\ldots b_i 0\ldots 0\times 2^n = \underline{\mathsf{z}}\\ 1 & \text{otherwise}\end{cases}
\end{aligned}$$

Definition 4.2 can easily be extended to intervals with denormalized numbers and intervals composed of negative floating-point values. But it is useless to extend it with intervals containing both negative and positive values as such an interval contains 0 and the property is not applicable. As a result, we got formulas that compute filtering similar to those resulting from the Marre and Michel property. So, the filtering by maximum ULP presented here is similar to the filtering of [14] which is restricted to addition/subtraction operators.

D. Filtering by maximum ULP on the product

Fig.4 gives an intuitive view of ULP maximum over the floating-point multiplication. Let $\mathsf{z} = \mathsf{x} \otimes \mathsf{y}$ and suppose that $\mathsf{z} \in [v_\mathsf{z}, v_\mathsf{z}]$ is a strictly positive constant, then there exists a greatest values x_m such that, its exists a value y, $x_m \otimes y = v_\mathsf{z}$. As for property 3.1, this value depends neither on x nor on y and do not necessarily belong to them. Let explain the existence of x_m on Fig.2. Consider x_m such that $x_m \otimes 0^+ = v_\mathsf{z}$[3], then by monotonicity of $\otimes$, $v_\mathsf{z} < x_m^+ \otimes 0^+$ (B in the figure). Thus, still by monotonicity, there is no strictly positive value y such that $x_m^+ \otimes y = v_\mathsf{z}$. By sign analysis, there is no negative or nil value y such that $x_m^+ \otimes y = v_\mathsf{z}$.

$\overline{\delta}_\otimes(z)$ is defined as $\overline{\delta}_\otimes(z) = |z| * 2^{p-min}$ and $\underline{\delta}_\otimes(z) = -\overline{\delta}_\otimes(z)$. The value of an interval z which maximizes $\overline{\delta}_\otimes$ (respecting minimizes $\underline{\delta}_\otimes$) is the one that has the greatest absolute value. Hence, for v_z an element of z (which does not contain 0) that maximizes $\overline{\delta}_\otimes$ then $\overline{\delta}_\otimes(v_\mathsf{z})$ (resp. $\underline{\delta}_\otimes(\mathsf{z})$) is an upper bound (resp. a lower bound) of x w.r.t. the constraint $\mathsf{z} = \mathsf{x} \otimes \mathsf{y}$. As the product is commutative, the same function can be used for y. Details can be found in [5].

As an example, consider the following problem:

example 4.2:

$$\begin{aligned}
&\mathsf{x} \in [-\mathit{INF}, +\mathit{INF}], \mathsf{y} \in [-\mathit{INF}, +\mathit{INF}],\\
&\mathsf{z} \in [0^+, 1.0\times 2^{-30}], \mathsf{z} = \mathsf{x} \otimes \mathsf{y}
\end{aligned}$$

We got $\overline{\delta}_\otimes(1.0\ldots0 \times 2^{-30}) = 1.0\ldots0 \times 2^{-30} * 2^{23-(-127)-1} = 1.0\ldots0 \times 2^{119}$ and the filtering of the domain of x and y yield:

$$\begin{aligned}
&\mathsf{x} \in [-1.0\ldots0\times 2^{119}, 1.0\ldots0\times 2^{119}],\\
&\mathsf{y} \in [-1.0\ldots0\times 2^{119}, 1.0\ldots0\times 2^{119}]
\end{aligned}$$

[3]this value always exists since this multiplication is equivalent to an exponent shifting

This filtering only applies when the value returned by $\overline{\delta}_\otimes$ is smaller than the greatest representable float distinct from $+\mathit{INF}$ (i.e., when $\mathsf{z} < 1\times 2^{-p}$). However, this filtering is useful when zero knowledge is available on the domains of x or y such as in the above example.

E. Filtering by maximum ULP on division

$\overline{\delta}_\oslash(z)$ is defined as $\overline{\delta}_\oslash(z) = |z| \otimes f_{max}$. For filtering x in constraint $\mathsf{z} = \mathsf{x} \oslash \mathsf{y}$, it suffices to take the value v_z that maximize $\overline{\delta}_\oslash$. This value is $\max(|\underline{\mathsf{z}}|, |\overline{\mathsf{z}}|)$. By considering $\underline{\delta}_\oslash(z) = -\overline{\delta}_\oslash(z)$, it is also possible to filter lower bounds.

F. Synthesis

In the previous sections, the following functions have been defined:

$$\begin{aligned}
&\overline{\delta}_\oplus(z) = z + 1.1\ldots1\times 2^{n+(p-i-1)} \text{ if } z = +1.z_1\ldots z_i 10\ldots0\times 2^n\\
&\overline{\delta}_\oplus(z) = 1.1\ldots1\times 2^{n+(p-i-1)} \qquad \text{if } z = -1.z_1\ldots z_i 10\ldots0\times 2^n\\
&\underline{\delta}_\oplus(z) = -\overline{\delta}_\oplus(-z)\\
&\overline{\delta}_\otimes(z) = |z| * 2^{p-min} \qquad \underline{\delta}_\otimes(z) = -\overline{\delta}_\otimes(z)\\
&\overline{\delta}_\oslash(z) = |z| \otimes f_{max} \quad \text{(for denormalized } z) \qquad \underline{\delta}_\oslash(z) = -\overline{\delta}_\oslash(z)
\end{aligned}$$

For each constraint, when $0 \notin \mathsf{z}$, interval consistency based filtering yields to

Constraint	$\mathsf{x} \subseteq$	$\mathsf{y} \subseteq$
$\mathsf{z} = \mathsf{x} \oplus \mathsf{y} (\mathsf{z} > 0)$	$[\ \underline{\delta}_\oplus(\zeta\ ),\ \ \overline{\delta}_\oplus(\zeta\ )]$	$[\ \underline{\delta}_\oplus(\zeta\ ),\ \ \overline{\delta}_\oplus(\zeta\ )]$
$\mathsf{z} = \mathsf{x} \oplus \mathsf{y} (\mathsf{z} < 0)$	$[-\overline{\delta}_\oplus(\zeta'), -\underline{\delta}_\oplus(\zeta')]$	$[-\overline{\delta}_\oplus(\zeta'), -\underline{\delta}_\oplus(\zeta')]$
$\mathsf{z} = \mathsf{x} \ominus \mathsf{y} (\mathsf{z} > 0)$	$[\ \underline{\delta}_\oplus(\zeta\ ),\ \ \overline{\delta}_\oplus(\zeta\ )]$	$[-\overline{\delta}_\oplus(\zeta\ ), -\underline{\delta}_\oplus(\zeta\ )]$
$\mathsf{z} = \mathsf{x} \ominus \mathsf{y} (\mathsf{z} < 0)$	$[-\overline{\delta}_\oplus(\zeta'), -\underline{\delta}_\oplus(\zeta')]$	$[\ \underline{\delta}_\oplus(\zeta'),\ \ \overline{\delta}_\oplus(\zeta')]$
$\mathsf{z} = \mathsf{x} \otimes \mathsf{y}$	$[\underline{\delta}_\otimes(M), \overline{\delta}_\otimes(M)]$	$[\underline{\delta}_\otimes(M), \overline{\delta}_\otimes(M)]$
$\mathsf{z} = \mathsf{x} \oslash \mathsf{y}$	$[\underline{\delta}_\oslash(M), \overline{\delta}_\oslash(M)]$	

with $\zeta = \text{destr}(\mathsf{z})$, $\zeta' = \text{destr}(-\mathsf{z})$ and $M = \max(|\underline{\mathsf{z}}|, |\overline{\mathsf{z}}|)$.

V. Conclusion

This paper is concerned with constraint solving over floating-point computations. Filtering consistency based algorithms are currently the privileged technology for attacking this problem. This paper reformulates the Marre and Michel property proposed in [14] for improving the filtering capabilities of the addition and subtraction operators. It also proposes to generalize the property to the case of multiplication and division, something that has not been proposed elsewhere. Further work includes the exploration of other properties based on linearization of floating-point computations, such as those proposed in [19].

References

[1] P.E. Ammann and J.C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.

[2] Andrea Arcuri. Theoretical analysis of local search in software testing. In *Proceedings of the 5th international conference on Stochastic algorithms: foundations and applications*, SAGA'09, pages 156–168, 2009.

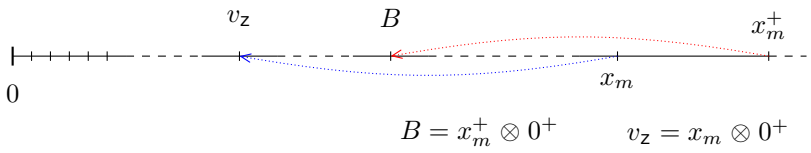

Figure 4. Existence of a limit such that $\forall y, x \otimes y \neq v_z$

[3] B. Blanc, F. Bouquet, A. Gotlieb, B. Jeannet, T. Jeron, B. Legeard, B. Marre, C. Michel, and M. Rueher. The v3f project. In *Proc. of the 1st workshop Constraints in Software Testing, Verification and Analysis (CSTVA'06), co-located with CP'06*, Nantes, France, Sep. 2006.

[4] B. Botella, A. Gotlieb, and C. Michel. Symbolic execution of floating-point computations. *The Software Testing, Verification and Reliability journal*, 16(2):pp 97–121, June 2006.

[5] Matthieu Carlier and Arnaud Gotlieb. Anr u3cat project report – wp1: Floating-point computations. Technical report, INRIA, Jan. 2011.

[6] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of Symp. on Principles of Programming Languages*, pages 238–252. ACM, 1977.

[7] P. Godefroid, N. Klarlund, and K. Sen. Dart: directed automated random testing. In *Proc. of PLDI'05*, pages 213–223, 2005.

[8] A. Gotlieb and B. Botella. Automated metamorphic testing. In *27th IEEE Annual International Computer Software and Applications Conference (COMPSAC'03)*, Dallas, TX, USA, November 2003.

[9] E. Goubault. Static analyses of the precision of floating-point operations. In *Static Analysis Symposium (SAS'01) and also in LNCS 2126*, pages 234–245, Paris, FR, July 2001.

[10] IEEE-754. Standard for binary floating-point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, February 1985.

[11] B. Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, 16(8):870–879, Aug. 1990.

[12] Roberto Castañeda Lozano, Christian Schulte, and Lars Wahlberg. Testing continuous double auctions with a constraint-based oracle. In *Principles and Practice of Constraint Programming - CP 2010 St. Andrews, Scotland, UK, Sep. 6-10, 2010.*, number 6308 in LNCS, pages 613–627, 2010.

[13] Bruno Marre and Benjamin Blanc. Test selection strategies for lustre descriptions in gatel. *Electronic Notes in Theoretical Computer Science*, 111:93 – 111, 2005.

[14] Bruno Marre and Claude Michel. Improving the floating point addition and subtraction constraints. In *Principles and Practice of Constraint Programming - CP'2010*, volume 6308 of *LNCS*, pages 360–367. 2010.

[15] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[16] C. Michel. Exact projection functions for floating point number constraints. In *Seventh Int. Symp. on Artificial Intelligence and MAthematics (7th AIMA)*, Fort Lauderdale, FL, USA, Jan. 2002.

[17] C. Michel, M. Rueher, and Y. Lebbah. Solving constraints over floating-point numbers. In *Proceedings of Principles and Practices of Constraint Programming (CP'01)*, Springer Verlag, LNCS 2239, pages 524–538, Paphos, Cyprus, November 2001.

[18] W. Miller and D. Spooner. Automatic generation of floating-point test data. *IEEE Transactions on Software Engineering*, 2(3):223–226, September 1976.

[19] Michel Rueher Mohammed Said Belaid, Claude Michel. Approximating floating-point operations to verify numerical programs. In *14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'10)*, ENS Lyon, France, Sep. 2010.

[20] Enyi Tang, Earl Barr, Xuandong Li, and Zhendong Su. Perturbing numerical calculations for statistical analysis of floating-point program (in)stability. In *Proc. of the 19th Int. Symp. on Software Testing and Analysis*, ISSTA '10, pages 131–142, 2010.

[21] E. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4), 1982.