Solving Range Constraints for Binary Floating-Point Instructions

Abraham Ziv, Merav Aharoni, Sigal Asaf
IBM Research Labs
Haifa University, Mount Carmel
Haifa 31905, ISRAEL
E-mail address: ziv@il.ibm.com

Abstract

We present algorithms that solve the following problem: given three ranges of floating-point numbers R_x, R_y, R_z, a floating-point operation (op), and a rounding-mode ($round$), generate three floating-point numbers $\bar{x}$, $\bar{y}$, $\bar{z}$ such that $\bar{x} \in R_x$, $\bar{y} \in R_y$, $\bar{z} \in R_z$, and $\bar{z} = round(\bar{x}$ *op* $\bar{y})$. This problem, although quite simple when dealing with intervals of real numbers, is much more complex when considering ranges of machine numbers. We provide full solutions for add and subtract, and partial solutions for multiply and divide. We use range constraints on the input operands and on the result operand of floating-point instructions to target corner cases when generating test cases for use in verification of floating-point hardware. The algorithms have been implemented in a floating-point test-generator and are currently being used to verify floating-point units of several processors.

1 Introduction

Floating-point unit verification presents a unique challenge in the field of processor verification. The particular complexity of this area stems from the vast test-space, which includes many corner cases that should each be targeted, and from the intricacies of the implementation of floating-point operations. Verification by simulation involves executing a subset of tests which is assumed to be a representative sample of the entire test-space [1]. In doing so, we would like to be able to define a particular subspace, which we consider as ”interesting” in terms of verification, and then generate tests selected at random out of the subspace.

Consider the following cases as examples of this approach. It is interesting to check the instruction FP-SUB, where the two inputs are normalized numbers and the output is denormal, since this means the inputs had values that were close to each other, resulting in a massive cancellation of the most significant bits of the inputs and underflow in the result.

As a more general example, consider the following 10 basic ranges of floating-point numbers: ±normalized numbers, ±denormal numbers, ±zero, ±infinity, and ±NaNs. For a binary instruction, it is interesting to generate test cases (when possible) for all the combinations of selecting two inputs and the output out of these ranges (1000 combinations).

Subsets of floating-point numbers may be represented in various ways. Some examples of common representations are:

1. Ranges of floating-point numbers.

2. Masks. In this representation each bit may take on one of the values 0, 1, X, where X specifies that both 0 and 1 are possible. For example *01X1X* represents the set {01010, 01011, 01110, 01111}.

3. Number of 1s (0s). This defines the set of all floating-point numbers that have a given number of 1s (0s).

Probably the most natural of these, in the context of test-generation, is the representation as ranges. The primary advantage in representing sets of numbers as ranges, is its simplicity. All the basic types of floating-point numbers are readily defined in terms of ranges. Many interesting sets of numbers can be efficiently represented as a union of ranges. Test generation for the FP-ADD instruction, where the input sets are described as masks, is discussed in [2]. Examples of other algorithms that may be used for test generation may be found in [3], [4], and [5].

In this paper, we describe algorithms that provide random solutions given constraints on the instructions add, subtract, multiply, and divide when the two inputs and the output sets are all represented as ranges. In other words, given a floating-point operation $op \in \{add, subtract, multiply, divide\}$, a rounding mode *round* $\in$ { *to-zero*, *to-positive-infinity*, *to-negative-infinity*, *to-nearest* }, and three ranges R_x, R_y, R_z of floating point-numbers, the algorithm will produce three random floating-point numbers, $\bar{x}$, $\bar{y}$, $\bar{z}$, such that $\bar{x} \in R_x$, $\bar{y} \in R_y$, $\bar{z} \in R_z$, and $\bar{z} = round(\bar{x}$ *op* $\bar{y})$.

The difficulty in solving range constraints stems from the fact that although a floating-point range appears to be a continuous set of numbers, each range is in fact a set of discrete machine numbers. To illustrate this, consider two floating-point numbers $\bar{x} \in R_x$ and $\bar{z} \in R_z$. It is quite possible that there exists a real number y, in the real interval defined by the endpoints of R_y, for which $round(\bar{x} \times y) = \bar{z}$, but that there does not exist such a machine number.

The algorithms presented have been implemented in a floating-point test-generator, FPgen, in the IBM Haifa Research Labs [6]. FPgen is an automatic floating-point test-generator, which receives as input the description of a floating-point coverage task [1] and outputs a random test that covers this task. A coverage task is defined by specifying a floating-point instruction and a set of constraints on the inputs, on the intermediate result(s), and on the final result. For each task, the generator produces a random test that satisfies all the constraints. FPgen employs various algorithms, both analytic and heuristic, to solve the various constraint types. Solving the general constraint problem is NP-Complete. When the constraints are given as ranges, FPgen employs the algorithms described in this paper in order to generate the test cases.

In Section, 2 we describe the problem more formally and outline our general approach. In Section 3, we present the continuous algorithm, which works efficiently for all four arithmetic operations in most cases. In Section 4, we investigate the cases in which the algorithm does not work efficiently for multiplication. In Section 5, we present a second algorithm for addition, which solves the cases that are too difficult for the continuous algorithm. Section 6 includes a summary of the results and some suggestions for future work in this area.

2 Problem Definition

2.1 Formalization

We investigate the following problem:

Given six machine numbers $\bar{a}_x$, $\bar{b}_x$, $\bar{a}_y$, $\bar{b}_y$, $\bar{a}_z$, $\bar{b}_z$, find an algorithm that produces two machine numbers $\bar{x}$, $\bar{y}$ such that $\bar{a}_x \leq \bar{x} \leq \bar{b}_x$, $\bar{a}_y \leq \bar{y} \leq \bar{b}_y$, and such that $\bar{z} = round(\bar{x}$ op $\bar{y})$ satisfies $\bar{a}_z \leq \bar{z} \leq \bar{b}_z$ where op $\in \{+, -, \times, \div\}$.

$round$ represents a rounding mode. It may be, for instance, one of the IEEE standard 754 [7] rounding modes: up, down, toward zero, to nearest/even.

Throughout the paper, we represent machine numbers using lower-case letters with an upper bar, for example $\bar{x}$. This notation is used to distinguish them from real numbers, which are represented by regular lower-case letters, for example x.

2.2 General algorithm

In the following sections, we present two algorithms, the continuous algorithm and the discrete algorithm. The continuous algorithm is based on the similarity between the problem discussed here, and the problem of finding such a solution when the ranges are real number intervals. In the case of real numbers, the problem is relatively simple: we reduce the intervals and find a solution by using the inverse of the required operation. For machine numbers, it is not possible to use the inverse operation, therefore the algorithm that we propose differs slightly. As we will show later, the continuous algorithm is very efficient for most cases, for all four operations. Howerever, there are certain cases for which this is not true.

The discrete algorithm is specific for addition and subtraction. It is slower than the continuous algorithm, however, if a solution exists, this algorithm will find it. Therefore, for addition, we suggest first running the continuous algorithm. If the algorithm does not terminate within a reasonable amount of time, use the discrete algorithm.

In general, our discussion focuses on a solution for IEEE 754 sets of floating-point numbers. However, the algorithms can easily be converted to non-IEEE rounding modes (such as round half down), and most of the results are valid for any floating-point format (such as decimal floating-point [8]).

Some of the discussion which follows is valid for all four arithmetic operations. However, an important part of the discussion is restricted to addition, subtraction, and multiplication. Note, the problems for addition and subtraction are actually equivalent, because $\bar{y}$ may be replaced by $-\bar{y}$. Hence subtraction is not discussed further. Throughout the paper, we indicate which parts are relevant to all operations and which parts are true only for specific operations.

2.3 Notation

- **Floating-point number representation.** Using the notation of the IEEE standard 754 [7], we represent a floating-point number v by three values: $S(v)$, $E(v)$, and $F(v)$, which denote the *sign*, the *exponent* and the *significand*, respectively. The value of v is

$$v = (-1)^{S(v)} \cdot 2^{E(v)} \cdot F(v)$$

- **p (precision)**. This signifies the number of bits in the significand, F, of the floating-point number.

- **ulp(x) or unit in the last place**. This is the weight of one unit in the least significant bit of the significand of x. The ulp of a floating-point number with exponent E and precision p, is 2^{E+1-p}.

3 The Continuous Algorithm

In the following section, we describe the continuous algorithm for addition. Similar algorithms are also valid for the division and multiplication problems. For the divide continuous algorithm, the symbols $+$ and $-$ should be replaced by the symbols $\div$ and $\times$, respectively. For the multiply continuous algorithm, the symbols $+$ and $-$ should be replaced by the symbols $\times$ and $\div$, respectively.

3.1 Eliminating the $round$ operation

We define the set

$$I'_z = \{z \mid \bar{a}_z \le round(z) \le \bar{b}_z\}.$$

Obviously I'_z is an interval, and $[\bar{a}_z, \bar{b}_z] \subset I'_z$. We denote the ends of this interval by a'_z, and b'_z.

Clearly $a'_z \le \bar{a}_z \le \bar{b}_z \le b'_z$, $a'_z < b'_z$, and I'_z is one of the intervals (a'_z, b'_z), $[a'_z, b'_z)$, $(a'_z, b'_z]$, $[a'_z, b'_z]$, depending on $\bar{a}_z$, $\bar{b}_z$ and on the rounding mode. Our problem can now be formulated as follows:

Find two machine numbers $\bar{x}$, $\bar{y}$ such that $\bar{a}_x \le \bar{x} \le \bar{b}_x$, $\bar{a}_y \le \bar{y} \le \bar{b}_y$, and $\bar{x} + \bar{y} \in I'_z$.

We define the set of all real number pairs that solve the above inequalities as follows

$$A_{x,y} = \{(x, y) \mid \bar{a}_x \le x \le \bar{b}_x,\ \bar{a}_y \le y \le \bar{b}_y,\ x + y \in I'_z\} \tag{1}$$

Clearly, the set of all solutions to our problem is the subset of $A_{x,y}$, which includes all the pairs in which x and y are machine numbers. In Figure 1, we depict the set of solutions for addition.

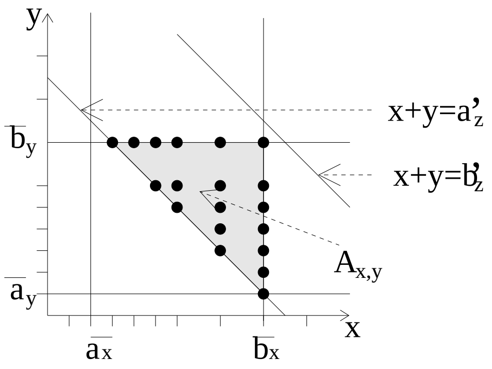

Figure 1. Solution set $A_{x,y}$ for addition

3.2 Reducing the input intervals

There are cases in which some of the intervals $[\bar{a}_x, \bar{b}_x]$, and $[\bar{a}_y, \bar{b}_y]$, I'_z can be reduced to smaller intervals, such that the problems remain equivalent. By *equivalent* we mean that the set $A_{x,y}$ is the same for both problems. We perform the reduction as follows:

For the $+$ operation and $I'_z = [a'_z, b'_z]$, and for every $(x, y) \in A_{x,y}$:

$$\bar{a}_x \le x \le \bar{b}_x,\ \bar{a}_y \le y \le \bar{b}_y,\ a'_z \le x + y \le b'_z.$$

In other words,

$$a'_z - y \le x \le b'_z - y,\ a'_z - x \le y \le b'_z - x.$$

The two relations above imply that

$$a'_z - \bar{b}_y \le x \le b'_z - \bar{a}_y,\ a'_z - \bar{b}_x \le y \le b'_z - \bar{a}_x,$$

Combining this with the original boundaries on x and y from Equation 1, we get new boundaries on x, y, and $x+y$, which we denote by $A_x, B_x, A_y, B_y, A_z, B_z$:

$$A_x = \max\{\bar{a}_x, a'_z - \bar{b}_y\} \le x \le \min\{\bar{b}_x, b'_z - \bar{a}_y\} = B_x$$

$$A_y = \max\{\bar{a}_y, a'_z - \bar{b}_x\} \le y \le \min\{\bar{b}_y, b'_z - \bar{a}_x\} = B_y$$

$$A_z = \max\{a'_z, \bar{a}_x + \bar{a}_y\} \le x + y \le \min\{b'_z, \bar{b}_x + \bar{b}_y\} = B_z.$$

We assumed that $I'_z = [a'_z, b'_z]$. However, if we have $I'_z = (a'_z, b'_z]$ or $[a'_z, b'_z)$ or (a'_z, b'_z), some of the $\le$ relations need to be replaced by $<$ relations. Apart from this, the results would be identical. We can summarize the results in the following way:

For the $+$ operation we have,

$$A_{x,y} = \{(x, y) \mid x \in I_x,\ y \in I_y,\ z = x + y \in I_z\}$$

where I_x, I_y, I_z are intervals (each end point of which is either closed or open) with the end points: A_x, B_x, A_y, B_y, A_z, and B_z, as described above.

In order to follow a similar argument for the $\times$ and the $\div$ operations, we have to first assume (with no loss of generality) that the operands are all positive.

Proposition 1 *The intervals I_x, I_y, I_z cannot be reduced any further.*

To see this, it suffices to show that for every $A_x < x < B_x$, there exists a y, such that $(x, y) \in A_{x,y}$ and likewise for every $A_y < y < B_y$ and for every $A_z < z < B_z$. We shall prove this for every $A_x < x < B_x$, with the $+$ operation. The rest of the cases have similar proofs.

Proof: Consider any x, such that $A_x < x < B_x$. From the definitions of A_x, B_x we deduce that $a'_z - \bar{b}_y < x < b'_z - \bar{a}_y$.

Consider now the expression $z - y$. The left hand side of the above double inequality is equal to $z - y$ with $z = a'_z$, $y = \bar{b}_y$. The right hand side is equal to the same expression with $z = b'_z$, $y = \bar{a}_y$. Therefore, by changing (y, z)

continuously, from $(a_z', \bar{b}_y)$ to $(b_z', \bar{a}_y)$, we deduce the existence of intermediate values of y and z, such that $z - y = x$ (or $z = x + y$) exactly. For these values of y and z, we have, $\bar{a}_y < y < \bar{b}_y$, $a_z' < x + y < b_z'$ (and obviously $\bar{a}_x < x < \bar{b}_x$), therefore $(x, y) \in A_{x,y}$. ∎

3.3 Finding a random solution using the continuous algorithm

The above discussion enables us to generate random solutions in the following way:

(i) Choose a random machine number $\bar{x} \in I_x$. If no such $\bar{x}$ exists, there is no solution. Stop.

(ii) Choose a random machine number $\bar{y}$ in the interval

$$[\max\{\bar{a}_y, a_z' - \bar{x}\}\ ,\ \min\{\bar{b}_y, b_z' - \bar{x}\}].$$

If no such $\bar{y}$ exists, go back to step (i).

(iii) If $\bar{a}_z \leq round(\bar{x} + \bar{y}) \leq \bar{b}_z$, return the result $(\bar{x}, \bar{y})$ and stop. Otherwise, go back to step (i).

Intuitively, we see that this method is very efficient if the projections of the machine number solutions onto the x, y and z axes give sets which are each dense in I_x, I_y and I_z respectively. At the other extreme, when the density of the machine numbers is relatively small, the algorithm may work for an unacceptably long amount of time. In the following two sections we discuss this problem in more depth.

4 Efficiency of the Continuous Algorithm for Multiplication

For the multiply operation, the algorithm of the previous section is adequate, except for one extreme case, when $\bar{a}_z = \bar{b}_z$.

We illustrate this using an example with binary floating-point numbers with a three bit significand:

$\bar{a}_x = (1.00)_2$, $\bar{b}_x = (1.01)_2$, $\bar{a}_y = \bar{b}_y = (1.11)_2$, $\bar{a}_z = \bar{b}_z = (10.0)_2$ with the round-up mode. Then,

$$round(\bar{x}\bar{y}) \in \{(1.11)_2, (10.1)_2\}$$

so there is no solution (step (i) cannot discover this) although $A_{x,y}$ is not empty. In this particular case, the continuous algorithm may run for an unreasonably long time.

The following theorem proves the sufficiency of the algorithm for all other cases. Note, this theorem is valid only for binary floating point systems.

Theorem 1 *For the $\times$ operation in a binary floating point system, assume that the set $A_{x,y}$ is not empty, and that $\bar{a}_z < \bar{b}_z$. Let $\bar{x}$ be a machine number in I_x and let $\bar{a}_y$ be a normalized machine number. Then there exists at least one machine number, $\bar{y}$, such that $(\bar{x}, \bar{y})$ is a solution.*

Note, the condition $\bar{a}_z < \bar{b}_z$ implies that there exist at least two machine numbers in I_z'. Therefore, the only case not covered by the theorem is the case where I_z' contains a single machine number.

The fact that $\bar{x} \in I_x$ means that there exists some $y_0 \in I_y$ (not necessarily a machine number), such that $(\bar{x}, y_0) \in A_{x,y}$.

The assumption that $\bar{a}_y$ is normalized is not really restrictive. This is because if both $\bar{x}$, $\bar{y}$ are denormals, their rounded product has only two possible values. Namely, 0 and the smallest positive denormalized machine number, depending on the rounding mode. This is a simple case, which can be easily treated. Therefore, in interesting cases, either $\bar{a}_x$ or $\bar{a}_y$ must be normalized.

Proof: We divide the discussion into three sub cases.

(i) $\bar{a}_y > a_z'/\bar{x}$, (ii) $\bar{b}_y < b_z'/\bar{x}$, (iii) $\bar{a}_y \leq a_z'/\bar{x}$, $\bar{b}_y \geq b_z'/\bar{x}$.

As mentioned above, there exists a real number, y_0, such that $(\bar{x}, y_0) \in A_{x,y}$. Obviously $\bar{a}_x \leq \bar{x} \leq \bar{b}_x$ in all three cases. Now, in case (i), we choose $\bar{y} = \bar{a}_y$. Consequently, $\bar{a}_y \leq \bar{y} \leq \bar{b}_y$ and

$$\bar{y} > a_z'/\bar{x} \quad \Leftrightarrow \quad \bar{x}\bar{y} > a_z'.$$

Because $(\bar{x}, y_0) \in A_{x,y}$ we must have $\bar{x}y_0 \leq b_z'$. However, (in case (i)) the point $(\bar{x}, \bar{a}_y) = (\bar{x}, \bar{y})$ lies on the lower boundary of $A_{x,y}$, so, $\bar{y} \leq y_0$, which means that either $\bar{y} = y_0$, hence $(\bar{x}, \bar{y}) \in A_{x,y}$, or $\bar{y} < y_0$, hence $\bar{x}\bar{y} < \bar{x}y_0 \leq b_z'$ and again $(\bar{x}, \bar{y}) \in A_{x,y}$.

In case (ii) we choose $\bar{y} = \bar{b}_y$, therefore, $\bar{a}_y \leq \bar{y} \leq \bar{b}_y$ and

$$\bar{y} < b_z'/\bar{x} \quad \Leftrightarrow \quad \bar{x}\bar{y} < b_z'.$$

Since $(\bar{x}, y_0) \in A_{x,y}$ we must have $\bar{x}y_0 \geq a_z'$. However (in case (ii)), the point $(\bar{x}, \bar{b}_y) = (\bar{x}, \bar{y})$ lies on the upper boundary of $A_{x,y}$, so, $\bar{y} \geq y_0$ and either $\bar{y} = y_0$, hence $(\bar{x}, \bar{y}) \in A_{x,y}$, or $\bar{y} > y_0$, hence $\bar{x}\bar{y} > \bar{x}y_0 \geq a_z'$ and again $(\bar{x}, \bar{y}) \in A_{x,y}$.

In case (iii), all points $(\bar{x}, y)$ between the points $(\bar{x}, a_z'/\bar{x})$, $(\bar{x}, b_z'/\bar{x})$ (inclusive) must satisfy $\bar{a}_y \leq y \leq \bar{b}_y$. Hence, in order to complete the proof, it suffices to show that there exists at least one machine number, $\bar{y}$, such that $a_z'/\bar{x} < \bar{y} < b_z'/\bar{x}$:

Let $\bar{y}_1$ be the largest machine number that satisfies $\bar{y}_1 \leq a_z'/\bar{x}$. We choose $\bar{y} = \bar{y}_1 + ulp(\bar{y}_1)$. Obviously $\bar{y}$ is a machine number and $\bar{y} > a_z'/\bar{x}$. We complete the proof by showing that $\bar{y} < b_z'/\bar{x}$. We denote $z_1 = \bar{x}\bar{y}_1$, $z = \bar{x}\bar{y}$. Then,

$$z - z_1 = \bar{x}(\bar{y}_1 + ulp(\bar{y}_1)) - \bar{x}\bar{y}_1 = \bar{x}\, ulp(\bar{y}_1).$$

Now we divide $z - z_1$ by $ulp(z_1)$. Using the notation from section 2.3,

$$\frac{z - z_1}{ulp(z_1)} = \bar{x}\frac{ulp(\bar{y}_1)}{ulp(z_1)} = \frac{z_1}{\bar{y}_1}\frac{ulp(\bar{y}_1)}{ulp(z_1)} =$$

$$\frac{2^{E(z_1)}S(z_1)}{2^{E(\bar{y}_1)}S(\bar{y}_1)}\,\frac{2^{E(\bar{y}_1)+1-p}}{2^{E(z_1)+1-p}} = \frac{S(z_1)}{S(\bar{y}_1)}.$$

$\bar{a}_y$ is a machine number satisfying $\bar{a}_y \leq a'_z/\bar{x}$. Therefore, $\bar{y}_1 \geq \bar{a}_y$, and $\bar{y}_1$ is normalized. Hence $S(\bar{y}_1) \geq 1$. Additionally, $0 < S(z_1) < 2$, so, $S(z_1)/S(\bar{y}_1) < 2$. We have, then, $z - z_1 < 2\,ulp(z_1)$. In addition $b'_z - a'_z \geq 2\,ulp(a'_z)$ because there exist at least two machine numbers in I'_z. This implies that $z < z_1 + 2\,ulp(z_1)$ and that $a'_z + 2\,ulp(a'_z) \leq b'_z$. Therefore,

$$z < z_1 + 2\,ulp(z_1) = \bar{x}\bar{y}_1 + 2\,ulp(\bar{x}\bar{y}_1) \leq$$

$$\bar{x}a'_z/\bar{x} + 2\,ulp(\bar{x}a'_z/\bar{x}) = a'_z + 2\,ulp(a'_z) \leq b'_z.$$

Hence,

$$\bar{x}\bar{y} < b'_z \quad \Leftrightarrow \quad \bar{y} < b'_z/\bar{x}$$

and the proof is complete. ▮

5 The Discrete Algorithm for the $+$ Operation

5.1 Preliminary discussion

There are some cases in which the algorithm of Section 3.3, applied to the $+$ operation, does not work efficiently. This occurs, for example, when there is a significant cancellation of bits; in other words, when the exponent of the result is smaller than the input exponents. We illustrate this point by an example using binary floating-point numbers with a three bit significand:

$\bar{a}_x = (1.00)_2$, $\bar{b}_x = (1.11)_2$, $\bar{a}_y = -(1.11)_2$, $\bar{b}_y = -(1.00)_2$, $\bar{a}_z = (0.00101)_2$, $\bar{b}_z = (0.00111)_2$, with round to nearest mode. Clearly the set $A_{x,y}$ in this example, is not empty. Yet, there is no solution of machine numbers. The set of solutions is sparse on the z axis, in the vicinity of the interval $[\bar{a}_z, \bar{b}_z]$. In general, the algorithm cannot detect whether such a solution exists.

Therefore, we need to find a different algorithm for addition, which solves these difficult cases as well. The focus of this algorithm will be to search for a solution within a subspace of the (x, y) plane. More specifically, the algorithm described below works only in cases where the rectangle $R = [\bar{a}_x, \bar{b}_x] \times [\bar{a}_y, \bar{b}_y]$ has the following properties:

(I) *The step size, Δy, between successive machine numbers along the y-axis, is constant throughout the rectangle*.

(II) *The step size, Δx, between successive machine numbers along the x-axis, may vary within the rectangle, but is always at least the size of Δy.*

The conditions on x and y are interchangeable.

In order to make use of this algorithm, the original rectangle R, which contains all the solution pairs, must be divided into sub-rectangles, each of which has these properties. A method for efficient partitioning of the original rectangle is described in Section 5.5.

In the most extreme case, the original rectangle may include all of the pairs of floating-point numbers (both positive and negative). In such a case, the number of subrectangles will be approximately $8(E_{\max} - E_{\min})$, which is close to 16000 for IEEE double precision. This algorithm is therefore slower than the one described in Section 3.3. Hence we recommend using first the continuous algorithm, and if it does not find a solution within a reasonable amount of time, to then use the discrete algorithm.

We first set up the theoretical framework underlying the algorithm, in Sections 5.2 and 5.3, by showing a necessary and sufficient condition for the existence of a solution. The algorithm itself is presented in Sections 5.4 and 5.5.

5.2 Interval lemmas

Lemma 1 *Given two intervals $[a_1, b_1]$, $[a_2, b_2]$, the condition $[a_1, b_1] \cap [a_2, b_2] \neq \emptyset$ is equivalent to $a_1 \leq b_1$, $a_2 \leq b_2$, $a_1 \leq b_2$, $a_2 \leq b_1$.*

This lemma has been formulated for closed intervals. We may similarly formulate this lemma for intervals in which one, or both, ends are open. The proof of the lemma is immediate.

Lemma 2 *Let P be a set of points on the real axis, let I be an interval on the real axis, and let I_P be a closed interval on the real axis, whose end points are both in P. Then, $I_P \cap I \cap P \neq \emptyset$ if and only if $I_P \cap I \neq \emptyset$ and $I \cap P \neq \emptyset$.*

Proof: It is easy to see that

$$I_P \cap I \cap P \neq \emptyset \Rightarrow \; I_P \cap I \neq \emptyset\,,\, I \cap P \neq \emptyset.$$

Now, assume that $I_P \cap I \neq \emptyset$ and that $I \cap P \neq \emptyset$. If at least one of the ends of I_P is in I, this end point is included in $I_P \cap I \cap P$. Therefore, we assume that both ends are outside of I. There are three possibilities: (i) both ends lie to the left of I, (ii) both ends lie to the right of I and (iii) the left end lies to the left of I and the right end lies to the right of I. In cases (i)and (ii) we have $I_P \cap I = \emptyset$, which contradicts our assumptions. In case (iii) we have $I \subseteq I_P$, so $I_P \cap I \cap P = I \cap P \neq \emptyset$.

▮

5.3 Condition for the existence of a solution

In what follows, we assume that the rectangle R satisfies conditions (I) and (II) of sub-section 5.1. Let $\bar{x}$ be a machine number in $[\bar{a}_x, \bar{b}_x]$. We define the set of points

$$D(\bar{x}) = \{\bar{x} + \bar{y} \mid \bar{y} \in [\bar{a}_y, \bar{b}_y],\; \bar{y} \; is \; a \; machine \; number\}.$$

Because of conditions (I) and (II), every point of $D(\bar{x})$ can be written in the form $\bar{a}_x + \bar{a}_y + k\,\Delta y$, where k is an integer. We also define $I(\bar{x})$ to be the smallest interval of real numbers that includes $D(\bar{x})$. Namely,

$$I(\bar{x}) = [\bar{x} + \bar{a}_y, \bar{x} + \bar{b}_y].$$

We will need the following lemma:

Lemma 3 *Assume that at least one point of the sequence $\bar{a}_x + \bar{a}_y + k\,\Delta y$ ($k = 0, \pm 1, \pm 2, \cdots$) is included in I'_z. Given a machine number $\bar{x} \in [\bar{a}_x, \bar{b}_x]$, a necessary and sufficient condition for the existence of a solution in $D(\bar{x})$ (i.e., $D(\bar{x}) \cap I'_z \neq \emptyset$) is the condition $I(\bar{x}) \cap I'_z \neq \emptyset$.*

Proof: We use Lemma 2 and substitute

$$P = \{\bar{a}_x + \bar{a}_y + k\,\Delta y \mid k = 0, \pm 1, \pm 2, \cdots\},$$

$$I = I'_z, \;\; I_P = I(\bar{x}).$$

Obviously $D(\bar{x}) = I(\bar{x}) \cap P$. Hence, from Lemma 2 we find that

$$I_P \cap I \cap P = I(\bar{x}) \cap I'_z \cap P =$$

$$D(\bar{x}) \cap I'_z \neq \emptyset \;\Leftrightarrow\; I(\bar{x}) \cap I'_z \neq \emptyset, \; I'_z \cap P \neq \emptyset$$

Because we assumed that $I'_z \cap P \neq \emptyset$, the condition $I(\bar{x}) \cap I'_z \neq \emptyset$ is equivalent to $D(\bar{x}) \cap I'_z \neq \emptyset$. ▮

Theorem 2 *Assume that I'_z is closed on both ends and that (I) and (II) are satisfied. A necessary and sufficient condition for the existence of a solution for the $+$ operation, is: At least one integer k, satisfies*

$$(a'_z - \bar{a}_x - \bar{a}_y)/\Delta y \leq k \leq (b'_z - \bar{a}_x - \bar{a}_y)/\Delta y$$

and at least one machine number $\bar{x}$, satisfies

$$\bar{a}_x \leq \bar{x} \leq \bar{b}_x \;,\; a'_z - \bar{b}_y \leq \bar{x} \leq b'_z - \bar{a}_y. \tag{2}$$

If the conditions are satisfied (i.e., there exist solutions), for each $\bar{x}$ that satisfies equation 2, there exist solutions of the form $(\bar{x}, \bar{y})$, where $\bar{x} + \bar{y}$ is in $D(\bar{x})$. For each machine number $\bar{x}$, which is not of this type, there are no solutions. All solutions with $\bar{x}$ are of the form

$$(\bar{x}, \bar{a}_y + n\,\Delta y)$$

where

$$(a'_z - \bar{x} - \bar{a}_y)/\Delta y \leq n \leq (b'_z - \bar{x} - \bar{a}_y)/\Delta y$$

$$n = 0, 1, \cdots, N \;,\; N = (\bar{b}_y - \bar{a}_y)/\Delta y.$$

Remark: Theorem 2 was formulated with the assumption that I'_z is closed at both of its ends. If this is not so, the formulation is similar when using the relation $<$ to replace some of the corresponding $\leq$ relations.

Proof: A number in $D(\bar{x}) \cap I'_z$ must be of the form $\bar{x} + \bar{y} = \bar{a}_x + \bar{a}_y + k\,\Delta y$ and must be included in I'_z. Therefore, from Lemma 3 we see that necessary and sufficient conditions for the existence of a solution are $a'_z \leq \bar{a}_x + \bar{a}_y + k\,\Delta y \leq b'_z$ for some k and $I(\bar{x}) \cap I'_z \neq \emptyset$. The first of these conditions is equivalent to

$$(a'_z - \bar{a}_x - \bar{a}_y)/\Delta y \leq k \leq (b'_z - \bar{a}_x - \bar{a}_y)/\Delta y$$

and the second is equivalent to $\bar{x} + \bar{a}_y \leq b'_z$, $\bar{x} + \bar{b}_y \geq a'_z$ from Lemma 1, which is equivalent to

$$a'_z - \bar{b}_y \leq \bar{x} \leq b'_z - \bar{a}_y.$$

The solutions which correspond to $\bar{x}$, must clearly be of the form $(\bar{x}, \bar{a}_y + n\,\Delta y)$, with $n = 0, 1, \cdots, N$, and they must satisfy $a'_z \leq \bar{x} + \bar{a}_y + n\,\Delta y \leq b'_z$. Therefore, we must have

$$(a'_z - \bar{x} - \bar{a}_y)/\Delta y \leq n \leq (b'_z - \bar{x} - \bar{a}_y)/\Delta y,$$

$$n = 0, 1, \cdots, N.$$

▮

5.4 The discrete algorithm

Theorem 2 leads us to formulate a second algorithm for the addition. Because the theorem assumes conditions (I) and (II), one must first divide the rectangle R into sub-rectangles, each of which either satisfies these conditions, or satisfies the conditions with x and y interchanged.

The following algorithm must be applied to each of these sub-rectangles in a random order, until a solution is found.

(i) Test for the existence of an integer k, which satisfies

$$(a'_z - \bar{a}_x - \bar{a}_y)/\Delta y \leq k \leq (b'_z - \bar{a}_x - \bar{a}_y)/\Delta y.$$

If no such k exists, stop. There is no solution.

(ii) Choose, at random, a machine number $\bar{x}$ that satisfies

$$\bar{a}_x \leq \bar{x} \leq \bar{b}_x \;,\; a'_z - \bar{b}_y \leq \bar{x} \leq b'_z - \bar{a}_y$$

(i.e. $\bar{x} \in I_x$). If no such $\bar{x}$ exists, stop. There is no solution.

(iii) Choose at random, an integer n that satisfies

$$(a'_z - \bar{x} - \bar{a}_y)/\Delta y \leq n \leq (b'_z - \bar{x} - \bar{a}_y)/\Delta y$$

$$n \in \{0, 1, \cdots, N\}$$

(such an n must exist!).
Return the solution $(\bar{x}, \bar{a}_y + n\,\Delta y)$ and stop.

The remark that follows Theorem 2, applies to this algorithm as well. We need only apply this algorithm to those sub-rectangles that intersect the sub-plane $a'_z \leq x + y \leq b'_z$.

5.5 Partitioning the original ranges into sub-rectangles

Several different methods can be used to partition the original rectangle into sub-rectangles. In the simplest method, each sub-rectangle includes exactly one exponent for both x and y. Using this method, in the worst case, we get approximately $4(E_{\max} - E_{\min})^2$ rectangles, which is around 16,000,000 for double precision. Therefore, this solution is impractical.

The following method generates, in the worst case, approximately $8(E_{\max} - E_{\min})$ sub-rectangles, which is around 16,000 for double precision. For each exponent, E, we define the following two rectangles:

1. The exponent of y is E and the exponent of $x \geq E$.

2. The exponent of x is E and the exponent of $y > E$.

For each quadrant, there are about $2(E_{\max} - E_{\min})$ rectangles. Because, in the worst case, we must define these rectangles for each quadrant of the plane, in total we get about $8(E_{\max} - E_{\min})$.

Figure 2 illustrates the partition to sub-rectangles for the first quadrant.

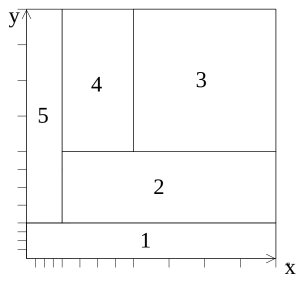

Figure 2. Dividing into rectangles

6 Summary and Future Work

We presented algorithms for addition, multiplication, and division, based on the fact the machine number ranges behave similarly to real number intervals in many cases. We pointed out that these algorithms are usually efficient and fast, and that similar algorithms may be easily formulated for other arithmetic operations.

We showed that the continuous algorithm is efficient for multiplication, whenever the result range consists of more than one value (i.e., when $\bar{a}_z \neq \bar{b}_z$). For addition, we presented the discrete algorithm, which always finds a solution, if one exists. This algorithm is slower than the continuous algorithm, and its use is recommended only in those cases in which the continuous algorithm does not work efficiently.

A natural extension of this work is to generalize the algorithm to fully support the $\div$ operation. This includes checking the effectiveness of the continuous algorithm and possibly finding an additional algorithm that solves those cases for which the continuous algorithm is insufficient. The solution for multiplication is not yet complete. We need to find an algorithm that efficiently solves the case of a single value in the result range.

It is also interesting to investigate the multiply-add instruction ($a \times b + c$) [9], which is under a standardization process, and which is known to be bug- prone. Solving this instruction obviously involves the problems that arise when solving the $+$ instruction. However, this problem is even more complex, as we have a variable step size between consecutive machine numbers in the product ($a \times b$).

References

[1] Raanan Grinwald, Eran Harel, Michael Orgad, Shmuel Ur, and Avi Ziv. User defined coverage - a tool supported methodology for design verification. In *Proc. 38th, Design Autometation Conference (DAC38)*, pages 158 – 163, 1998.

[2] Abraham Ziv and Laurent Fournier. Solving the generalized mask constraint for test generation of binary floating point add operation. *Theoretical Computer Science (to appear)*.

[3] W. Kahan. A test for correctly rounded sqrt. http://www.cs.berkeley.edu/∼wkahan/SQRTest.ps.

[4] Lee D. McFearin and David W. Matula. Generation and analysis of hard to round cases for binary floating point division. In *IEEE Symposium on Computer Arithmetic (ARITH15)*, pages 119 – 127, 2001.

[5] Michael Parks. Number-theoretic test generation for directed rounding. In *Proc. IEEE 14th Symp. Computer-Arithmetic (ARITH14)*, pages 241 – 248, 1999.

[6] S. Asaf and L. Fournier. FPgen - a deep-knowledge test-generator for floating point verification. Technical Report H-0140, IBM Israel, 2002. http://domino.watson.ibm.com/library/cyberdig.nsf/Home.

[7] IEEE standard for binary floating point arithemetic, 1985. An American national standard, ANSI/IEEE Std 754.

[8] M. Cowlishaw, E. Schwarz, R. Smith, and C. Webb. A decimal floating-point specification. In *Proc. IEEE 15th Symp. Computer-Arithmetic (ARITH15)*, pages 147 – 154, 2001.

[9] W. Kahan. Lectures notes on the status of IEEE standard 754 for binary floating-point arithemetic., 1996. http://www.cs.berkeley.edu/∼wkahan/ieee754status/IEEE754.pdf.