Serdica J. Computing **4** (2010), 29–42

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A MATHEMATICAL BASIS FOR AN INTERVAL ARITHMETIC STANDARD

Gerd Bohlender, Ulrich Kulisch*

Abstract. Basic concepts for an interval arithmetic standard are discussed in the paper. Interval arithmetic deals with closed and connected sets of real numbers. Unlike floating-point arithmetic it is free of exceptions. A complete set of formulas to approximate real interval arithmetic on the computer is displayed in section 3 of the paper. The essential comparison relations and lattice operations are discussed in section 6. Evaluation of functions for interval arguments is studied in section 7. The desirability of variable length interval arithmetic is also discussed in the paper. The requirement to adapt the digital computer to the needs of interval arithmetic is as old as interval arithmetic. An obvious, simple possible solution is shown in section 8.

1. Introduction. Interval arithmetic [10, 1] has been used for many years in applications which require highly reliable results. In contrast with

ACM Computing Classification System (1998): G.1.0.

Key words: Interval arithmetic, IEEE standardization, variable precision interval arithmetic, evaluation of functions, hardware support.

*The authors thank Goetz Alefeld, Klaus Braune, Reinhard Kirchner, Rudolf Lohner, and Markus Neher for valuable discussions.

floating-point arithmetic which only delivers approximations of mathematical results, a correctly implemented and applied interval arithmetic over the floating-point numbers always computes an enclosure of the corresponding exact mathematical results. This makes it possible to prove mathematical results in a rigorous way on the computer.

Interval arithmetic can be realized via IEEE 754 floating-point arithmetic, but this leads to an unacceptable loss of efficiency, in particular because switching of rounding modes is extremely time consuming, and because case distinctions for multiplication and division must be programmed in software. With very little extra hardware, interval arithmetic can be made as fast as floating-point arithmetic.

Basing on a proposal by IFIP WG 2.5 [9] and one of the authors [7], the setting up of an IEEE standardization group on interval arithmetic was planned at a seminar in Dagstuhl castle, Germany, January 2008. The intent was to create a standard for interval arithmetic. This standardization group was authorized by IEEE on June 11, 2008 and set up as IEEE group P1788 on July 16, 2008. Since then, work is making progress and many details have been worked out [2]. Work and voting is organized in "Motions".

This paper is intended as a mathematical background for the detailed technical work on the future standard.

2. Interval Sets and Mappings. Interval arithmetic over the real numbers deals with closed and connected sets of the real numbers $\mathbb{R}$. Here an interval is denoted by an ordered pair. The first element is the lower bound and the second is the upper bound. The lower bound shall not be greater than the upper bound. If an interval is bounded it is written as $[a, b]$, with $a, b \in \mathbb{R}$. If it is unbounded it is written as $(-\infty, a]$ or $[b, +\infty)$ with $a, b \in \mathbb{R}$ or $(-\infty, +\infty)$ where the parentheses indicate that the bounds $-\infty$ and $+\infty$ are not elements of the interval. The set of all such bounded and unbounded intervals including the empty set is denoted by $\overline{\mathbb{IR}}$. With respect to set inclusion as an order relation $\{\overline{\mathbb{IR}}, \subseteq\}$ is a complete lattice. It is bounded from below by the empty set $\emptyset$ and from above by the set $(-\infty, +\infty)$. Bold face will be used for intervals denoted by a single letter. The lower bound is denoted by a subscript 1 and the upper bound is denoted by a subscript 2.

Arithmetic for real numbers as well as for sets of real numbers is well defined. For interval operands the result of an operation always leads to an interval again and the bounds of the result can be given by simple expressions for the bounds of the operands. This gives interval arithmetic the right to exist. The corresponding formulas can be deduced from the definition of the operations for sets of real numbers in a strict mathematical manner [6, 7]. The calculus $\{\overline{\mathbb{IR}}, +, -, *, /\}$ is free of exceptions.

On the computer real numbers are approximated by the subset of floating-point numbers as defined by the IEEE 754 floating-point arithmetic standard, for instance. The set of all floating-point numbers is denoted by $\mathbb{F}$. The subset of all bounded or unbounded intervals of $\overline{\mathbb{IR}}$ with finite bounds of $\mathbb{F}$ including the empty set is denoted by $\overline{\mathbb{IF}}$. Intervals of $\overline{\mathbb{IR}}$, arithmetic operations, and comparison relations for these are approximated by intervals, arithmetic operations, and comparison relations for intervals of the set $\overline{\mathbb{IF}}$.

We consider here only the set of double precision binary or decimal floating-point numbers. For other floating-point formats and encodings the considerations are similar.

A real number or an interval over the real numbers is mapped onto the smallest floating-point interval that contains the number or interval respectively. This mapping $\Diamond\ \overline{\mathbb{IR}} \to \overline{\mathbb{IF}}$ is characterized by the following properties:

(R1) $\Diamond\ \boldsymbol{a} = \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IF}}$,

(R2) $\boldsymbol{a} \subseteq \boldsymbol{b} \Rightarrow \Diamond\ \boldsymbol{a} \subseteq \Diamond\ \boldsymbol{b}$, for $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IR}}$,

(R3) $\boldsymbol{a} \subseteq \Diamond\ \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IR}}$,

(R4) $\Diamond\ (-\boldsymbol{a}) = -\Diamond\ \boldsymbol{a}$, for all $\boldsymbol{a} \in \overline{\mathbb{IR}}$.

3. Arithmetic Operations for Intervals. The IEEE floating-point arithmetic standard 754 specifies arithmetic with four roundings: to the nearest floating-point number, downwards, upwards, and towards zero. For these operations the following notations will be used:
$+, -, *, /$ for the operations with rounding to the nearest floating-point number,
$\overset{+}{\triangledown}, \overset{-}{\triangledown}, \overset{*}{\triangledown}, \overset{/}{\triangledown}$ for the operations with rounding downwards,
$\overset{+}{\triangle}, \overset{-}{\triangle}, \overset{*}{\triangle}, \overset{/}{\triangle}$ for the operations with rounding upwards,[1] and
$*|, -|, +|, /|$ for the operations with rounding towards zero (chopping).[2]

[1]In our Pascal extension (available since 1980) and the Fortran extension we developed for and with IBM (available 1990) pairs of keybord symbols $+<, -<, *<, /<$ and $+>, ->, *>, />$ have been used for the operations with rounding downwards and upwards, respectively.

[2]Frequently used programming languages do not allow four plus, minus, multiply, and divide operators for floating-point numbers. A future interval arithmetic standard could or should specify names for low level operations with the directed roundings. They could be: *addp*, *subp*, *mulp*, *divp*, *addn*, *subn*, *muln*, and *divn*. Here p stands for rounding towards positive and n for rounding towards negative. With these operations interval routines would be fully transferable from one processor to another.

With these notations for bounded intervals $\boldsymbol{a} = [a_1, a_2], \boldsymbol{b} = [b_1, b_2] \in \overline{\mathbb{IF}}$ the following interval arithmetic operations $+, -, *,$ and $/$ are defined:

Table 1. Definition of arithmetic operations for intervals

Addition	$[a_1, a_2] + [b_1, b_2] = [a_1 \overset{+}{\triangledown} b_1, a_2 \overset{+}{\triangle} b_2]$.
Subtraction	$[a_1, a_2] - [b_1, b_2] = [a_1 \overset{-}{\triangledown} b_2, a_2 \overset{-}{\triangle} b_1]$.

Multiplication $[a_1, a_2] * [b_1, b_2]$	$[b_1, b_2]$ $b_2 \leq 0$	$[b_1, b_2]$ $b_1 < 0 < b_2$	$[b_1, b_2]$ $b_1 \geq 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \overset{*}{\triangledown} b_2, a_1 \overset{*}{\triangle} b_1]$	$[a_1 \overset{*}{\triangledown} b_2, a_1 \overset{*}{\triangle} b_1]$	$[a_1 \overset{*}{\triangledown} b_2, a_2 \overset{*}{\triangle} b_1]$
$a_1 < 0 < a_2$	$[a_2 \overset{*}{\triangledown} b_1, a_1 \overset{*}{\triangle} b_1]$	$[min(a_1 \overset{*}{\triangledown} b_2, a_2 \overset{*}{\triangledown} b_1),$ $max(a_1 \overset{*}{\triangle} b_1, a_2 \overset{*}{\triangle} b_2)]$	$[a_1 \overset{*}{\triangledown} b_2, a_2 \overset{*}{\triangle} b_2]$
$[a_1, a_2], a_1 \geq 0$	$[a_2 \overset{*}{\triangledown} b_1, a_1 \overset{*}{\triangle} b_2]$	$[a_2 \overset{*}{\triangledown} b_1, a_2 \overset{*}{\triangle} b_2]$	$[a_1 \overset{*}{\triangledown} b_1, a_2 \overset{*}{\triangle} b_2]$

Division, $0 \notin \boldsymbol{b}$ $[a_1, a_2]/[b_1, b_2]$	$[b_1, b_2]$ $b_2 < 0$	$[b_1, b_2]$ $b_1 > 0$
$[a_1, a_2], a_2 \leq 0$	$[a_2 \overset{/}{\triangledown} b_1, a_1 \overset{/}{\triangle} b_2]$	$[a_1 \overset{/}{\triangledown} b_1, a_2 \overset{/}{\triangle} b_2]$
$[a_1, a_2], a_1 < 0 < a_2$	$[a_2 \overset{/}{\triangledown} b_2, a_1 \overset{/}{\triangle} b_2]$	$[a_1 \overset{/}{\triangledown} b_1, a_2 \overset{/}{\triangle} b_1]$
$[a_1, a_2], a_1 \geq 0$	$[a_2 \overset{/}{\triangledown} b_2, a_1 \overset{/}{\triangle} b_1]$	$[a_1 \overset{/}{\triangledown} b_2, a_2 \overset{/}{\triangle} b_1]$

Division, $0 \in \boldsymbol{b}$ $[a_1, a_2]/[b_1, b_2]$	$\boldsymbol{b} =$ $[0, 0]$	$[b_1, b_2]$ $b_1 < b_2 = 0$	$[b_1, b_2]$ $0 = b_1 < b_2$
$[a_1, a_2], a_2 < 0$	$\emptyset$	$[a_2 \overset{/}{\triangledown} b_1, +\infty)$	$(-\infty, a_2 \overset{/}{\triangle} b_2]$
$[a_1, a_2], a_1 \leq 0 \leq a_2$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[a_1, a_2], a_1 > 0$	$\emptyset$	$(-\infty, a_1 \overset{/}{\triangle} b_1]$	$[a_1 \overset{/}{\triangledown} b_2, +\infty)$

Division by an interval that includes zero in the last table (i.e. an interval having zero end-point(s))) leads to unbounded intervals. To be complete, arithmetic operations for unbounded intervals also have to be defined now.

The first rule is that any operation with the empty set $\emptyset$ has the empty set as its result.

Arithmetic operations for unbounded intervals of $\overline{\mathbb{IF}}$ can be performed on the computer by using the above formulas for bounded intervals if in addition a few formal rules for operations with $-\infty$ and $+\infty$ are applied. These rules are shown in the following tables. A dash in the table means that a corresponding operation needs not be defined. It does not occur in the interval operations.

Table 2. Formal rules for operations with $+\infty$ and $-\infty$

Addition	$-\infty$	b	$+\infty$
$-\infty$	$-\infty$	$-\infty$	$-$
a	$-\infty$	$-$	$+\infty$
$+\infty$	$-$	$+\infty$	$+\infty$

Subtraction	$-\infty$	b	$+\infty$
$-\infty$	$-$	$-\infty$	$-\infty$
a	$+\infty$	$-$	$-\infty$
$+\infty$	$+\infty$	$+\infty$	$-$

Multiplication	$-\infty$	$b < 0$	0	$b > 0$	$+\infty$
$-\infty$	$+\infty$	$+\infty$	0	$-\infty$	$-\infty$
$a < 0$	$+\infty$	$-$	$-$	$-$	$-\infty$
0	0	$-$	$-$	$-$	0
$a > 0$	$-\infty$	$-$	$-$	$-$	$+\infty$
$+\infty$	$-\infty$	$-\infty$	0	$+\infty$	$+\infty$

Division	$-\infty$	$+\infty$
a	0	0

These rules are not new in principle. They are well established in real analysis and IEEE 754 provides them anyway. The only rule that goes beyond IEEE 754 is

$$0 * (-\infty) = (-\infty) * 0 = 0 * (+\infty) = (+\infty) * 0 = 0. \tag{1}$$

This rule follows quite naturally from the definition of unbounded intervals as sets of real numbers. However, it should not be taken as a new mathematical law. It is just a shortcut to easily compute the bounds of the result of an operation on unbounded intervals.

With the mapping $\diamondsuit : \overline{\mathbb{IR}} \to \overline{\mathbb{IF}}$ and its properties listed at the end of section 1 the operations defined in this section have the following property which defines them uniquely:

(RG) $\boldsymbol{a} \circledcirc \boldsymbol{b} := \diamondsuit\, (\boldsymbol{a} \circ \boldsymbol{b})$, for all $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IF}}$ and all $\circ \in \{+, -, *, /\}$.

4. Remarks on the Arithmetic Operations.

I. In the table for division by an interval that includes zero the case $b_1 < 0 < b_2$ is missing. This needs some explanation.

A basic concept of mathematics is that of a function or mapping. A function consists of a pair (f, D_f). It maps each element x of its domain of definition D_f on a unique element y of the range R_f of f, $f : D_f \to R_f$.

In real analysis division by zero is not defined. Thus a rational function $y = f(x)$ where the denominator is zero for $x = c$ is not defined for $x = c$, i.e., c is not an element of the domain of definition D_f. Since the function $f(x)$ is not defined at $x = c$ it does not have any value or property there. In this strict mathematical sense, division by an interval $[b_1, b_2]$ with $b_1 < 0 < b_2$ is not well posed. For division the set $b_1 < 0 < b_2$ devolves into the two distinct sets $[b_1, 0]$[3] and $[0, b_2]$ and division by an interval $[b_1, b_2]$ with $b_1 < 0 < b_2$ actually consists of two divisions, the result of which again consists of two distinct sets. In each case the result is a single unbounded interval. The two divisions should be performed separately. Division by the two sets $[b_1, 0]$ and $[0, b_2]$ is shown in the relevant table.

The situation is plainly shown by the signs of the bounds of the divisor before the division is executed. For interval multiplication or division a case selection has to be done (by hardware or software) anyhow before the operations are performed. In the case $b_1 < 0 < b_2$ the sign of b_1 is negative and the sign of b_2 is positive.

In the user's program, however, the two divisions appear within a single operation, as division by an interval $[b_1, b_2]$ with $b_1 < 0 < b_2$. So an arithmetic operation in the user's program delivers two distinct results. This is an unusual situation in conventional computing.[4]

A solution to the problem would be for the computer to provide a flag for *distinct intervals*. The situation occurs if the divisor is an interval that contains zero as an interior point. In this case the flag would be raised and signaled to the user. The user may then apply a routine of his choice to deal with the situation as is appropriate for his application.

This routine could be: Modify the operands and recompute, or continue the computation with one of the sets and ignore the other one, or put one of the sets on a list and continue the computation with the other one, or return the entire set of real numbers $(-\infty, +\infty)$ as result and continue the computation, or stop computing, or any other action.

A somewhat natural solution would be to continue the computation on different tasks, one for each interval. But the situation can occur repeatedly. How many tasks would we need? Future multicore processors will provide a large number of units and perhaps allow to run many tasks in parallel. A similar

[3]Since division by zero does not contribute to the solution set it does not matter whether a round parenthesis or square bracket is used here.

[4]It would be very convenient for computing if other operations would also deliver two answers: floating-point addition and subtraction the rounded result and the error, multiplication the product to the double length and division the quotient and the remainder.

situation occurs in global optimization using subdivision. After a certain test several candidates may be left for further investigation.

Newton's method reaches its ultimate elegance and strength in the extended interval Newton method. It computes all (single) zeroes in a given domain. If a function has several zeroes in a given interval its derivative becomes zero in that interval also. Thus Newton's method applied to that interval delivers two distinct sets. This is how the extended interval Newton method separates different zeroes. If the method is continued along two separate paths, one for each of the distinct intervals it finally computes all zeroes in the given domain. If the method continues with only one of the two distinct sets and ignores the other one it computes an enclosure of only one zero of the given function. If the interval Newton method delivers the empty set, the method has proved that there is no zero in the initial interval.

II. If interval arithmetic is hardware supported then execution of the operations listed above is about as fast as execution of the corresponding floating-point operations. It is thus not reasonable to define and study operations between floating-point numbers and intervals in order to save computing time. Floating-point arithmetic and interval arithmetic are different calculi for approximate arithmetic for real numbers. They should be kept strictly separate.[5]

Of course, computing with result verification often makes use of floating-point computations. If executed in IEEE 754 arithmetic this may lead to exceptional results. So there remains the question of how results like $-\infty$, $+\infty$, NaN, -0, $+0$ can reasonably be mapped on floating-point intervals.

The following would be reasonable: -0 and $+0$ can only mean 0. Intervals are sets of real numbers. Since NaN is not a real number it should be mapped on the empty set and since $-\infty$ and $+\infty$ are also not real numbers their image could or should also be the empty set. If the image of the result of a floating-point computation is the empty set the user should be informed.

III. The empty set $\emptyset$ may occur as a result of an interval operation as listed in the tables of Section 3. The result of any operation with the empty set $\emptyset$ was defined to be the empty set. This suggests an encoding of the empty set in an IEEE environment by $\emptyset = [+NaN, -NaN]$. Then the rules for interval arithmetic listed in section 2 can also be applied to the empty set. By the well established rules of IEEE 754 for NaN an operation with the empty set would then automatically produce the empty set as the result.

[5]The XSC-languages allow real and interval data and operations between these in an expression. However, all real data are immediately interpreted as intervals and all operations are performed as interval operations.

The encoding $\emptyset = [+NaN, -NaN]$ for the empty set also turns out to be useful for the definition of comparison relations for intervals. These will be studied in section 6.

5. Variable Precision Interval Arithmetic. The success of interval arithmetic is based on two arithmetical features: One is double precision interval arithmetic. The other is variable precision interval arithmetic [10, 11, 14, 1]. For interval evaluation of an algorithm (a sequence of arithmetic operations) in the real number field a theorem by R. E. Moore [11] states that increasing the precision by k digits reduces the error bounds by b^{-k}, i.e., results can always be guaranteed to a number of correct digits by using variable precision interval arithmetic (for details see [1], [14]). Variable length interval arithmetic can be made very fast by a fast exact dot product and complete arithmetic [6], [8], [9].

An exact dot product for the double precision format is the basic tool to achieve high speed variable (dynamic) precision arithmetic for real and interval data. Pipelining gives it high speed, and exactitude brings very high accuracy into computation. There is no way to compute a dot product faster than the exact method. By pipelining, it can be computed in the time the processor needs to read the data, i.e., it comes with utmost speed [5, 6]. Variable length interval arithmetic fully benefits from such speed [6]. No software simulation can go as fast. With operator overloading variable length interval arithmetic is very easy to use.

6. Comparison Relations and Lattice Operations. Three comparison relations are important for intervals of $\overline{\mathbb{IF}}$:

(2) *equality*, *less than or equal*, and *set inclusion*.

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be intervals of $\overline{\mathbb{IF}}$ with bounds $a_1 \le a_2$ and $b_1 \le b_2$ respectively. Then the relations *equality* and *less than or equal* in $\overline{\mathbb{IF}}$ are defined by:

$$\boldsymbol{a} = \boldsymbol{b} \quad :\Leftrightarrow a_1 = b_1 \wedge a_2 = b_2,$$
$$\boldsymbol{a} \le \boldsymbol{b} \quad :\Leftrightarrow a_1 \le b_1 \wedge a_2 \le b_2.$$

Since bounds for intervals of $\overline{\mathbb{IF}}$ may be $-\infty$ or $+\infty$ all floating-point comparison relations in this section are executed as if performed in the lattice $\{\mathbb{F}^*, \le\}$ with $\mathbb{F}^* := \mathbb{F} \cup \{-\infty\} \cup \{+\infty\}$.

With the order relation $\leq$, $\{\overline{\mathbb{IF}}, \leq\}$ is a lattice. The *greatest lower bound* (glb) and the *least upper bound* (lub) of $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IF}}$ are the intervals

$$\begin{aligned} glb(\boldsymbol{a}, \boldsymbol{b}) &:= [min(a_1, b_1), min(a_2, b_2)], \\ lub(\boldsymbol{a}, \boldsymbol{b}) &:= [max(a_1, b_1), max(a_2, b_2)]. \end{aligned}$$

The greatest lower bound and the least upper bound of an interval with the empty set are both the empty set.

The inclusion relation in $\overline{\mathbb{IF}}$ is defined by

$$\boldsymbol{a} \subseteq \boldsymbol{b} :\Leftrightarrow b_1 \leq a_1 \wedge a_2 \leq b_2. \tag{3}$$

With the relation $\subseteq$, $\{\overline{\mathbb{IF}}, \subseteq\}$ is also a lattice. The least element in $\{\overline{\mathbb{IF}}, \subseteq\}$ is the empty set $\emptyset$ and the greatest element is the interval $(-\infty, +\infty)$. The infimum of two elements $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{IF}}$ is the intersection and the supremum is the interval hull (convex hull):

$$\begin{aligned} inf(\boldsymbol{a}, \boldsymbol{b}) = \quad & \boldsymbol{a} \cap \boldsymbol{b} &:= [max(a_1, b_1), min(a_2, b_2)] \quad \text{or the empty set } \emptyset, \\ sup(\boldsymbol{a}, \boldsymbol{b}) = \quad & \boldsymbol{a} \overline{\cup} \boldsymbol{b} &:= [min(a_1, b_1), max(a_2, b_2)]. \end{aligned}$$

The intersection of an interval with the empty set is the empty set. The interval hull with the empty set is the other operand.

If in the formulas for $glb(\boldsymbol{a}, \boldsymbol{b})$, $lub(\boldsymbol{a}, \boldsymbol{b})$, $\boldsymbol{a} \cap \boldsymbol{b}$, $\boldsymbol{a} \overline{\cup} \boldsymbol{b}$, a bound is $-\infty$ or $+\infty$ a parenthesis should be used for this interval bound to denote the resulting interval. This bound is not an element of the interval.

If in any of the comparison relations defined here both operands are the empty set, the result is true. If in (3) $\boldsymbol{a}$ is the empty set the result is true. Otherwise the result is false if in any of the three comparison relations only one operand is the empty set.[6]

A particular case of inclusion is the relation *element of.* It is defined by

$$a \in \boldsymbol{b} :\Leftrightarrow b_1 \leq a \wedge a \leq b_2.$$

Another useful check is whether $[a_1, a_2]$ is an interval at all, that is, if $a_1 \leq a_2$.

---

[6] A convenient encoding of the empty set in an IEEE environment may be $\emptyset = [+NaN, -NaN]$. Then most comparison relations and lattice operations considered in this section would deliver the correct answer if conventional rules for NaN are applied. However, if $\boldsymbol{a} = \emptyset$ then set inclusion (3) and computing the interval hull do not follow this rule. So in these two cases it must be checked whether $\boldsymbol{a} = \emptyset$ before the operations can be executed.

7. Evaluation of Functions. Let f be a function and D_f its domain of definition. For an interval $\boldsymbol{x} \subseteq D_f$, the range $range(f, \boldsymbol{x})$ of f is defined as the set of the function's values for all $x \in \boldsymbol{x}$:

$$range(f, \boldsymbol{x}) := \{f(x) | x \in \boldsymbol{x}\}.$$

On the computer, interval evaluation of a real function $f(x)$ for $\boldsymbol{x} \subseteq D_f$ should deliver a highly accurate enclosure of the range $range(f, \boldsymbol{x})$ of the function.

Evaluation of a function $f(x)$ for an interval $\boldsymbol{x}$ with $\boldsymbol{x} \cap D_f = \emptyset$, of course, does not make sense, since $f(x)$ is not defined for values outside its domain D_f. The empty set $\emptyset$ should be delivered and an error message may be given to the user.

There are, however, applications in interval arithmetic where information about a function f is useful when $\boldsymbol{x}$ exceeds the domain D_f of f. The interval $\boldsymbol{x}$ may also be the result of overestimation during an earlier interval computation.

In such cases the range of f can only be computed for the intersection $\boldsymbol{x}' := \boldsymbol{x} \cap D_f$:

$$range(f, \boldsymbol{x}') := range(f, \boldsymbol{x} \cap D_f) := \{f(x) | x \in \boldsymbol{x} \cap D_f\}.$$

To prevent the wrong conclusions being drawn, the user must be informed that the interval $\boldsymbol{x}$ had to be reduced to $\boldsymbol{x}' := \boldsymbol{x} \cap D_f$ to compute the delivered range. A particular flag for *domain overflow* may serve this purpose. An appropriate routine can be chosen and applied if this flag is raised. See also [12]. We give several examples:

Example 1: $l(x) := log(x)$, $D_{log} = (0, +\infty)$, $log((0, 2]) = (-\infty, log(2)]$. But also $log([-5, 2]') = log((0, 2]) = (-\infty, log(2)]$. The flag *domain overflow* should be set. It informs the user that the function has been evaluated for the intersection $\boldsymbol{x}' := \boldsymbol{x} \cap D_f = [-5, 2] \cap (0, +\infty) = (0, 2]$.

Example 2: $h(x) := sqrt(x)$, $D_{sqrt} = [0, +\infty)$,
$sqrt([1, 4]) = [1, 2]$, $sqrt([4, +\infty)) = [2, +\infty)$.
$sqrt([-5, -1]) = \emptyset$, an error message "$sqrt$ not defined for $[-5, -1]$" may be given to the user.
$sqrt([-5, 4]') = sqrt([0, 4]) = [0, 2]$. The flag *domain overflow* should be set. It informs the user that the function has been evaluated for the intersection $\boldsymbol{x}' := \boldsymbol{x} \cap D_f = [-5, 4] \cap [0, +\infty) = [0, 4]$.

Example 3: $k(x) := sqrt(x) - 1$, $D_k = [0, +\infty)$,
$k([-4, 1]') = k([0, 1]) = sqrt([0, 1]) - 1 = [-1, 0]$. The flag *domain overflow* should

be set. It informs the user that the function has been evaluated for the intersection $\boldsymbol{x}' := \boldsymbol{x} \cap D_f = [-4, 1] \cap [0, +\infty) = [0, 1]$.

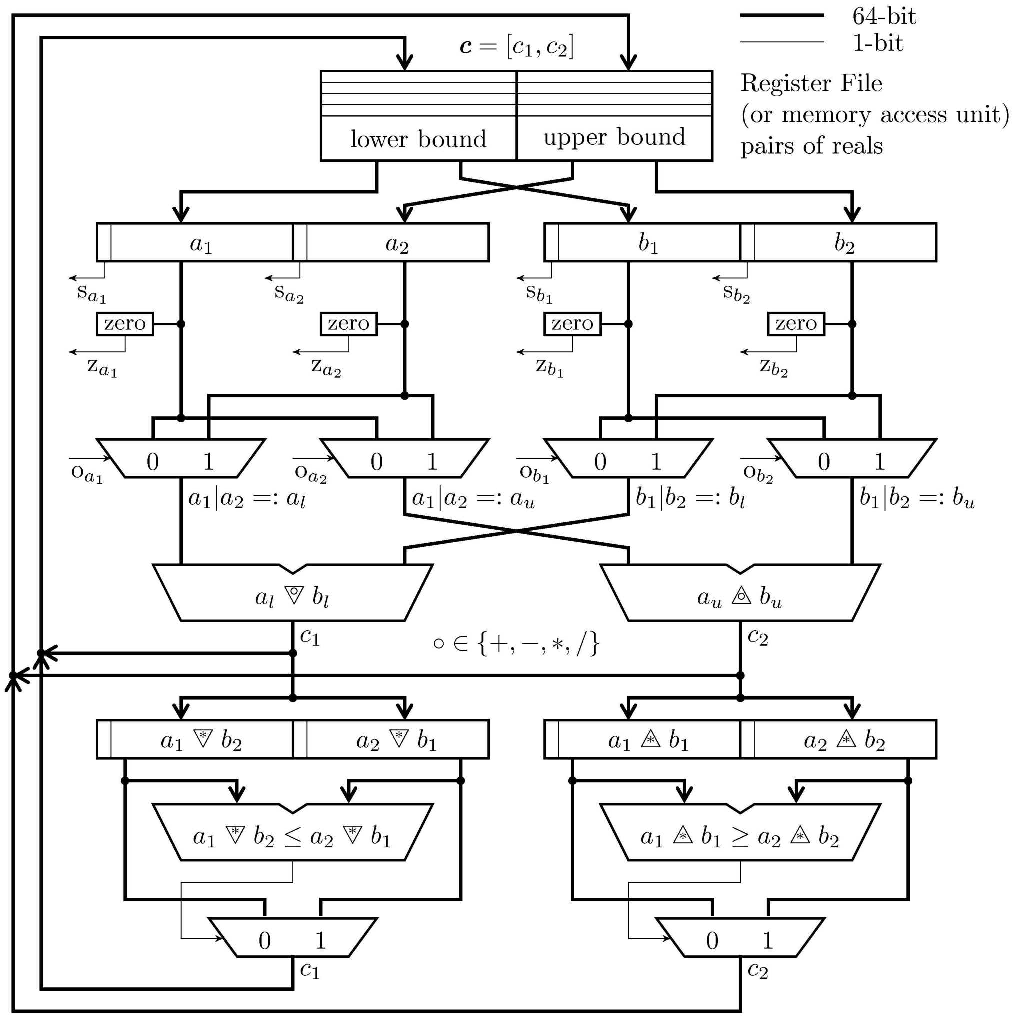
8. Hardware Support for Interval Arithmetic. In the early paper on interval arithmetic (1958) by Teruo Sunaga entitled *Theory of an Interval Algebra and its Application to Numerical Analysis* the last sentence states: *A future problem will be: To revise the structure of the automatic digital computer from the standpoint of interval calculus and topology.* So the requirement to adapt the digital computer to the needs of interval arithmetic is as old as interval arithmetic itself. A solution to the problem is not given in Sunaga's paper. At the time of the paper the technology was poor. There was no hope of getting it realized on computers in those days.

Figure 1 gives a brief sketch of what hardware support for interval arithmetic may look like. It would not be hard to realize it in modern technology. The circuitry broadly speaks for itself. The interval operands are loaded in parallel from a register file or a memory access unit. Then, after multiplexers have selected for the appropriate operands, the lower bound of the result is computed with rounding downwards and the upper bound with rounding upwards with the selected operands. In case of multiplication if both operands contain zero as an interior point a second multiplication is necessary. The result of both multiplications is forwarded to a comparison unit. Here for the lower bound of the result the lesser and for the upper bound the greater of the two products is selected. This lower part of the circuitry could also be used to perform comparison relations.

Table 3 shows the control signals for the operand selection by the multiplexers. These signals are computed from the signs of the bounds of the interval operands $\boldsymbol{a} = [a_1, a_2]$ and $\boldsymbol{b} = [b_1, b_2]$. In case of multiplication the signal $ms = s_{a1} \cdot \overline{s_{a2}} \cdot s_{b1} \cdot \overline{s_{b2}}$ is zero if only one product pair is to be computed, and it is one if a second product pair is to be computed. Every operand selector signal can be realized by two or three gates! For more details see [4] or [6].

Table 3. Operand selection signals

os	$+$	$-$	$*$	$/$
o_{a1}	0	0	$s_{b2} + \overline{s_{a1}} \cdot s_{b1} + ms$	$s_{b2} + s_{a1} \cdot s_{b1}$
o_{a2}	1	1	$\overline{s_{b1}} + \overline{s_{a1}} \cdot \overline{s_{b2}} + ms$	$\overline{s_{b1}} + s_{a2} \cdot \overline{s_{b2}}$
o_{b1}	0	1	$\overline{ms} \cdot (s_{a2} + s_{a1} \cdot \overline{s_{b2}})$	$\overline{s_{a1}} + \overline{s_{a2}} \cdot s_{b1}$
o_{b2}	1	0	$\overline{s_{a1}} + \overline{s_{a2}} \cdot \overline{s_{b1}} + ms$	$s_{a2} + s_{a1} \cdot s_{b1}$



operands: $\boldsymbol{a} = [a_1, a_2]$, $\boldsymbol{b} = [b_1, b_2]$, result: $\boldsymbol{c} = [c_1, c_2]$.
s: sign, z: zero, o: operand select.

Fig. 1. Circuitry for Interval Operations

The authors of this paper are convinced that hardware support for interval arithmetic is absolutely necessary. The simpler a standard for interval arithmetic is kept the more likely it is that it will result in hardware support for interval arithmetic.

REFERENCES

[1] Alefeld G., J. Herzberger. Introduction to Interval Computations. Academic Press, New York, 1983.

[2] IEEE Society IEEE Interval Standard Working Group – P1788. http://grouper.ieee.org/groups/1788/

[3] Kahan W. A More Complete Interval Arithmetic. Lecture Notes prepared for a summer course at the University of Michigan, June 17–21, 1968.

[4] Kirchner R., U. Kulisch. Hardware support for interval arithmetic. *Reliable Computing*, **12** (2006), No 3, 225–237.

[5] Kulisch U. W. Advanced Arithmetic for the Digital Computer – Design of Arithmetic Units. Springer-Verlag, Wien, New York, 2002.

[6] Kulisch U. W. Computer Arithmetic and Validity – Theory, Implementation and Applications. De Gruyter, Berlin, New York, 2008.

[7] Kulisch U. W. Complete Interval Arithmetic and its Implementation on the Computer. In: Numerical Validation in Current Hardware Architectures (Eds A. Cuyt, W. Krämer, W. Luther, P. Markstein), Springer-Verlag, Berlin – Heidelberg, 2009, 7–26.

[8] Lohner R. Interval Arithmetic in Staggered Correction Format. In: Scientific Computing with Automatic Result Verification (Eds. E. Adams, U. Kulisch), Academic Press, San Diego, 1993, 301–321.

[9] IFIPWG-IEEE754R. Letter of the IFIP WG 2.5 to the IEEE Computer Arithmetic Revision Group, 2007. http://www.mathematik.uni-karlsruhe.de/ianm2/∼kulisch.

[10] Moore R. E. Interval Analysis. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1966.

[11] Moore R. E. Methods and Applications of Interval Analysis. SIAM, Philadelphia, Pennsylvania, 1979.

[12] Popova E. Interval Operations Involving NaNs. *Reliable Computing*, **2** (1996), No 2, 161–165.

[13] Ratz D. On Extended Interval Arithmetic and Inclusion Isotony. Preprint, Institut für Angewandte Mathematik, Universität Karlsruhe, 1999.

[14] Rump S. M. Kleine Fehlerschranken bei Matrixproblemen. Dissertation, Universität Karlsruhe, 1980.

Gerd Bohlender
Institute for Applied and Numerical Mathematics 2
KIT – Karlsruhe Institute of Technology
Kaiserstraße 12, D-76131 Karlsruhe
Germany
e-mail: `gerd.bohlender@kit.edu`

Ulrich Kulisch
Institute for Applied and Numerical Mathematics 2
KIT – Karlsruhe Institute of Technology
Kaiserstraße 12, D-76131 Karlsruhe
Germany
e-mail: `ulrich.kulisch@kit.edu`

Received November 3, 2009
Final Accepted February 4, 2010