Annales Univ. Sci. Budapest., Sect. Comp. 40 (2013) 183–200

SYMBOLIC INTERVAL MANIPULATION

Sándor Czirbusz (ELTE, Hungary)

Dedicated to Professors Zoltán Daróczy and Imre Kátai on the occasion of their 75th birtday

Communicated by Antal Járai

(Received May 31, 2013; accepted June 16, 2013)

Abstract. When we are manipulating functions with computer programs, especially with computer algebra systems, we easily find ourselves confronted with the problem of modifying the range and the domain of a function with our program. If the underlying spaces are topological or even more, finite dimensional euclidean spaces, the domains and ranges are mostly topologically simple, i.e. closed, compact, or open sets. If the space is one dimensional, the simplest case is that, the sets are intervals. In numerical computations one of the classic methods is Interval Analysis or Interval Arithmetics, but this method uses closed intervals and is mainly interested in numerical accuracy. We care mostly about open sets and the numerical precision is not too important. In this article we treat this by modifying the method, to manipulate arbitrary intervals symbolically, and we solved this problem in the Computer Algebra system Maple®.

1. Introduction

In [4] we investigated the regularity properties of functional equations with the Maple computer algebra system. The usual algebraic manipulation of equations requires computing the domain of the substituted independent variables. These manipulations are often simple arithmetic operations and the domains

Key words and phrases: Interval arithmetics, symbolic computations.
2010 Mathematics Subject Classification: 65G40, 68W30.
https://doi.org/10.71352/ac.40.183

are simple real intervals. As a solution the application of interval arithmetics arises naturally, but the standard interval analysis uses only closed intervals, and in contrast we usually deal with open intervals. The necessity of using interval methods on arbitrary intervals was mentioned in [4]. As a simple example, let us consider the dilogarithm equation:

$$f(x) + f(y) + f(1-xy) + f\left(\frac{1-x}{1-xy}\right) + f\left(\frac{1-y}{1-xy}\right) = 0\,,$$

where $f :]0,1[\to \mathbb{R}$ and the variables are from the domain

$$\{(x,y) : 0 < x, y < 1\}\,.$$

We need to know the exact domain of the expressions in the brackets. When we are solving or transforming functional equations we often introduce new variables; in this case we must compute the domain of the new variables as well. Naturally there are complicated equations, however the inner expressions are usually rational arithmetic expressions.

2. Classic interval analysis at a glance

2.1. Arithmetics

The classic interval analysis is an efficient tool for numerical computations. In 1966 Ramon E. Moore published his first book [12], which remains a standard reference to this day. Ever since then, many books and articles were born, see for example [15, 13, 10, 7, 8, 6, 5]. The proofs of properties stated in this section can be found in almost all of them. The classic methods in essence can be used to solve our problems and we do not need more general methods, like directed intervals, modal intervals or affine arithmetics. In the classical sense by an interval we mean a compact connected subset of the real line, and use the so called *end–point notation* $X = [\underline{X}, \overline{X}]$, where $\underline{X} \leqq \overline{X}$ are real numbers. If the equality holds, we call it a *degenerated interval*, and it is straightforward to identify this interval with the real number $\underline{X} = \overline{X}$. First we need the basic arithmetic operators. If X, Y are intervals, then

$$X \odot Y = \{x \odot y : x \in X, y \in Y\}\,, \tag{2.1}$$

where $\odot \in \{-, +, \cdot, /\}$. Momentarily for the division we assume that $0 \notin Y$. In the next section we will discuss the case when the divisor interval contains 0.

Based on the above definition, we can define for example, the reciprocal of an interval:

$$1/Y = \{1/y : y \in Y\},$$

and we can define one-variable interval functions by

$$f(X) = \{f(x) : x \in X\},$$

if $f : \mathbb{R} \to \mathbb{R}$ is a function. In addition, we can use the set operations over intervals, but obviously the union of two intervals is not an interval if their intersection is empty. In this case sometimes we can use the *interval hull* of these intervals:

$$X \underline{\cup} Y = [\min(\underline{X}, \underline{Y}), \max(\overline{X}, \overline{Y})].$$

The interval addition and multiplication are commutative and associative; the degenerate intervals $[0, 0]$ and $[1, 1]$ are the additive and multiplicative identities. Generally, the inverses do not exist, but the addition satisfies the cancellation law. In additionally, instead of distributivity law we have only the *subdistributivity*:

$$X(Y + Z) \subset XY + XZ, \tag{2.2}$$

where X, Y, Z are intervals. This property is crucial in interval analysis, it serves as the basis of the later discussed fundamental theorem.

Now we summarize, how we can calculate the result intervals. Let $X = [\underline{X}, \overline{X}]$, $Y = [\underline{Y}, \overline{Y}]$ denote the operand intervals. Then it is easy to check that

$$\begin{aligned}
X + Y &= [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}], \\
-Y &= [-\overline{Y}, -\underline{Y}], \\
X - Y &= X + (-Y) = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}], \\
X \cdot Y &= [\min S, \max S], \text{where } S = \{\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}\}, \\
1/Y &= [1/\overline{Y}, 1/\underline{Y}], \text{if } 0 \notin Y.
\end{aligned}$$

The division is the multiplication with the reciprocal. Later we discuss the case of $0 \in Y$.

There is an other representation of intervals, the *midpoint–radius representation*. For this we need some simple and useful functions: the *midpoint* of the interval X is $m(X) = \frac{1}{2}(\underline{X} + \overline{X})$, the *width* is $w(X) = \overline{X} - \underline{X}$, the *absolute–value* is $|X| = \max(|\underline{X}|, |\overline{X}|)$. In this representation we can use the midpoint of the interval, and the radius, which is the half of the width.

2.2. Analysis

As mentioned after equation (2.1), this definition is suitable for defining the functions over the intervals. Naturally, we can generalize it for several variables, namely if f is a function over $\mathbb{R}^n$, and $X_1, \ldots, X_n$ ($n \in \mathbb{N}$) intervals, then

$$f(X_1, \ldots, X_n) = \{f(x_1, \ldots, x_n) : x_1 \in X_1, \ldots, x_n \in X_n\}\,.$$

This is the special case of the usual extension of a function. This function is sometimes called the *united extension* because it is the union of all the points mapped by the function from the domain. Generally it is not a trivial task to find this extension, but in some special cases it does not cause problems. For example, if $f : \mathbb{R} \to \mathbb{R}$ is monotone, the image of an interval X will be $[\min(f(\underline{X}), f(\overline{X})), \max(f(\underline{X}), f(\overline{X}))]$.

We would think that, if a function is defined via some formula the *natural extension* (substituting the real variable with an interval variable) the resulting function is the united extension of the original function. But this is not true, and the situation is even worse, see the $f(x) = x(1-x)$ simple real function as an example [14]. We get different results depending on the evaluation method of this expression. Let us substitute the real variable x with the interval variable $X = [0, 1]$. If we evaluate it in its original form, the result of $X \cdot (1 - X)$ is $[0, 1]$. But if we perform the multiplication first, that is we compute $X - X^2$, the result is $[-1, 1]$. So the resulting interval functions – obtained by substituting an interval in place of the real variable – are different. We note that even the square of an interval does not equal with the self multiplication.

We say that the function F is an *interval extension* of the real function f, if for all x $F([x, x]) = f(x)$, or more generally in case of several variables, if

$$F([x_1, x_1], \ldots, [x_n, x_n]) = f(x_1, \ldots, x_n)\,.$$

Usually, we can say that there is no unique interval extension of a function. It is easy to prove that interval arithmetic operators are satisfying the following property: if $Y_i \subset X_i, i = 1, 2$ then $Y_1 \odot Y_2 \subset X_1 \odot X_2$. This motivates the following:

Definition 2.1. The F interval function is *inclusion isotonic*, if from $Y_i \subset X_i$, $i = 1, \ldots, n, n \in \mathbb{N}$ follows that $F(Y_1, \ldots, Y_n) \subset F(X_1, \ldots, X_n)$.

An important class of functions fulfills the criterion of the above definition: the so–called *rational interval functions*, which can be computed with finite interval arithmetic steps.

The next theorem highlights the central role of united extension in interval arithmetics:

Theorem 2.1 (Fundamental Theorem of Interval Analysis)**.** *If function F is an inclusion isotonic interval extension of function f, then*

$$f(X_1, \ldots, X_n) \subset F(X_1, \ldots, X_n)$$

For the proof of this theorem see [14]. By this theorem, we can think about united extensions as the "smallest" extensions.

We can define distance on the set of all closed intervals:

$$d(X, Y) = \max(|\underline{X} - \underline{Y}|, |\overline{X} - \overline{Y}|) \tag{2.3}$$

This is a metric, the proof is a simple calculation. Sometimes this metric is called the *Moore–metric*. It is obvious that this distance is the well–known Hausdorff–distance, and therefore we know that the above metric space is complete, and the identification of the real numbers with the degenerated intervals is an isometric embedding. It is an easy task to prove that the interval sequence X_k converges to the interval X, if and only if the sequence of endpoints of the X_k intervals converge to the endpoints of X.

Definition 2.2. We say that the interval function F *Lipschitz* is on the interval X_0, if for every $X \subset X_0$ interval there exist an L nonnegative real number, for which

$$w(F(x)) \leqq Lw(X)\,, \tag{2.4}$$

where w is the width function.

It is easy to see the following

Facts:

1. *The natural interval extension of real rational functions is Lipschitz.*
2. *If a real-valued function f is Lipschitz on an interval X_0, then the united extension of f is a Lipschitz interval extension on X_0.*

2.3. Implementations

There are many implementations of interval arithmetics, they are used for numerical calculations. There is a comprehensive list of these methods at the Interval and Related Software website [3]. A lot of books and articles deal with some of these implementations, for example in [14] there are many examples programmed in the Intpak package of MatLab. Because we are interested in symbolic computations, we will talk about the Maple and Sage implementations.

Intervals in Maple

In the CAS Maple, the question had not been settled satisfactorily. Maple uses different approaches to implement interval arithmetics.

The INTERVAL object There is, an essentially undocumented feature in Maple, the *INTERVAL()* object. Faithful to the traditions of Maple, it is a polymorphic object. We can define with this object sequences of closed intervals in form $INTERVAL(a..b, c..d, \ldots)$ or bounded variables written in the form $INTERVAL(x, a..b)$. The bounds may be infinite values. This construction can be used as an argument or as the return value of the *evalr*() function. For example, if we compute the value $evalr\left(sin\left(INTERVAL\left(2..7\right)\right)\right)$, then the result is $INTERVAL\left(-1..\sin\left(2\right)\right)$. As another example let us see the expression $evalr(|x|)$. Maple evaluates it to

$$INTERVAL\left(INTERVAL\left(x, 0..\infty\right), -INTERVAL\left(x, -\infty..0\right)\right) .$$

There is an other function which belongs to this construction, the *shake*() function, which computes a bounding interval for a given value with a given accuracy. For example $shake(e, 5)$ is $INTERVAL(2.718010..2.718554)$.

The range object In the previous construction there is a "hidden" range construction: the".." type. This may be considered as an interval constant, but the usability of this construction is very limited. Its usage in Maple is twofold. First, if the endpoints are integers, we can use it as an iterator. The other application is to indicate the x and y ranges of drawings in plots.

The RealRange() function This construction is closest to our demands, with this function we can represent arbitrary intervals, because it uses the *Open*() function, which is part of the Maple's "assume" system. The documentation of this facility is very poor. There are no operators for this construction, and the inner automatic simplification makes it very circumstantial.

intpakX package This is an excellent package for Maple from the University of Wuppertal, but only for closed intervals and with numerical targets. See the program, examples and documentation in [9].

Intervals in SAGE

Sage supports arbitrary precision real (and complex) closed interval arithmetics for numerical computations. This is the SAGE implementation of the

MPFI library. It is somewhat surprising that the default representation is the so-called "question" style, that is, the notation 1.414213562373095? means that the preceding digit is possibly wrong by ± 1. For details see [1, 2].

3. Dealing with arbitrary intervals

Reviewing these concepts so far, the restriction to closed intervals is mostly unnecessary. The critical points are as follows:

Problems.

1. *The identification of degenerate open or half–open intervals with real numbers is impossible.*

2. *The Moore–distance will only be a quasimetric, so the space of arbitrary intervals is not a complete metric space.*

3. *The convergence of endpoints is problematic.*

4. *The inheritance of Lipschitz condition is not guaranteed.*

These problems are not too dangerous for us. Our main goal is to handle the rational functions, and in these cases the problems (2)-(4) are not significant. The first problem is more interesting. We can not identify arbitrary degenerate intervals with real numbers, because we get empty sets, if they are not closed. So the most convenient solution is to keep them as pairs of numbers.

About the operations

The definitions of interval arithmetic operations are independent from the type of the interval, they are algebraic operations between subsets of $\mathbb{R}$. Similar operations with sets are well known for example in convex geometry or group theory. In the case of intervals we can perform them simply on the number pairs, i.e. on endpoints. If we used only open intervals, there would be no problem, but it is not too hard to combine different interval types. The task to be solved is only to decide the type of the endpoints. The open endpoints are "greedy", they "consume" the closed endpoints. We will talk about this question more precisely and in more detail.
Hereafter we will say that an interval endpoint is closed or open weather that it belongs to the interval or not. Next we examine the interval arithmetic

operations in detail. If the real number x is an endpoint of some interval, then $Open(x) = true$, if x is an open endpoint and $Open(x) = false$, if x is a closed endpoint. The proposed and realized method can handle the unbounded intervals too. Next we summarize all of these rules.

Addition

As we have seen, if we add two intervals, X and Y, the endpoints of the result are $\underline{X} + \underline{Y}$ and $\overline{X} + \overline{Y}$ respectively. Naturally, if one of the endpoints is open than the resulting endpoint is open as well. More precisely, if $Z = X + Y$, then

$$Open(\underline{Z}) = Open(\underline{X}) \vee Open(\underline{Y})$$
$$Open(\overline{Z}) = Open(\overline{X}) \vee Open(\overline{Y}),$$

where $\vee$ is the logical disjunction.
The rules for addition, when there endpoints are not finite, can be found in table 1 on page 190. The letters a, b are arbitrary real numbers.

$+$	$-\infty$	a	∞
$-\infty$	$-\infty$	$-\infty$	undef.
b	$-\infty$	$a+b$	∞
∞	undef.	∞	∞

Table 1. Addition with infinite values

Subtraction

In this case as we have seen the calculation should be carried out with opposite endpoints respectively. Therefore if $Z = X - Y$ then $X - Y = X + (-Y) = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}]$ and

$$Open(\underline{Z}) = Open(\underline{X}) \vee Open(\overline{Y})$$
$$Open(\overline{Z}) = Open(\overline{X}) \vee Open(\underline{Y}).$$

In this case, the rules for calculating with infinite values are in Table 2.

Multiplication

The multiplication is relatively easy by the definition

$$Z = X \cdot Y = [\min S, \max S],$$

-	-∞	a	∞
$-\infty$	undef.	$-\infty$	$-\infty$
b	∞	$b-a$	$-\infty$
∞	∞	∞	undef.

Table 2. Subtraction with infinite values

where $S = \{\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}\}$. Although we do not care for the numerical efficiency much, we use the well known fact that we do not need four multiplications, see [14, 10]. The number of multiplications is three in the worst case. The details are shown in Table 3, when the zero is not in both of the intervals. In this situation we need only two multiplications.

Case	$\underline{Z}$	$\overline{Z}$
$0 \leqq \underline{X}$ and $0 \leqq \overline{Y}$	$\underline{X} \cdot \underline{Y}$	$\overline{X} \cdot \overline{Y}$
$\underline{X} < 0 < \overline{X}$ and $0 \leqq \underline{Y}$	$\underline{X} \cdot \overline{Y}$	$\overline{X} \cdot \overline{Y}$
$\overline{X} <\leqq 0$ and $0 \leqq \underline{Y}$	$\underline{X} \cdot \overline{Y}$	$\overline{X} \cdot \underline{Y}$
$0 \leqq \underline{X}$ and $\underline{Y} < 0 < \overline{Y}$	$\overline{X} \cdot \underline{Y}$	$\overline{X} \cdot \overline{Y}$
$\overline{X} < 0$ and $\underline{Y} < 0 < \overline{Y}$	$\underline{X} \cdot \overline{Y}$	$\underline{X} \cdot \underline{Y}$
$0 \leqq \underline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \overline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\overline{Y} \leqq 0$	$\overline{X} \cdot \underline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} \leqq 0$ and $\underline{Y} \leqq 0$	$\overline{X} \cdot \overline{Y}$	$\underline{X} \cdot \underline{Y}$
$\underline{X} < 0 < \overline{X}$ and $\underline{Y} < 0 < \overline{Y}$	See later	

Table 3. Cases of interval multiplication 1

In this case it is easy to calculate the values of the *Open*() function, because it is the logical or of *Open*() values of actually used endpoints.

When both of intervals contain the zero, we really need three multiplications. The calculation process is described in Table 4.

The calculation of *Open*() values for the endpoints is straightforward, but it is much easier when represented as program code than a mathematical expression. This will be shown in the *Realization* section. The only thing that remains is to see the multiplication with infinite values in Table 5. We note that here we deviate from the IEEE 754 standard. The standard suggest that the product of zero and an infinite value is undefined (not a number, NaN), but it seems that it is more suitable for our purposes to have zero as the result.

Case	$\underline{Z}$	$\overline{Z}$
$0 \leqq \lvert\underline{X}\rvert \leqq \overline{X}$ and $0 \leqq \lvert\underline{Y}\rvert \leqq \overline{Y}$	$\min\{\underline{X}\cdot\overline{Y}, \overline{X}\cdot\underline{Y}\}$	$\overline{X}\cdot\overline{Y}$
$0 \leqq \overline{X} \leqq \lvert\underline{X}\rvert$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\min\{\underline{X}\cdot\overline{Y}, \overline{X}\cdot\underline{Y}\}$	$\underline{X}\cdot\underline{Y}$
$0 \leqq \lvert\underline{X}\rvert \leqq \overline{X}$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\overline{X}\cdot\underline{Y}$	$\max\{\underline{X}\cdot\underline{Y}, \overline{X}\cdot\overline{Y}\}$
$0 \leqq \overline{X} \leqq \lvert\underline{X}\rvert$ and $0 \leqq \overline{Y} \leqq \lvert\underline{Y}\rvert$	$\underline{X}\cdot\overline{Y}$	$\max\{\underline{X}\cdot\underline{Y}, \overline{X}\cdot\overline{Y}\}$

Table 4. Cases of interval multiplication 2

$\cdot$	$-\infty$	$b<0$	0	$b>0$	∞
$-\infty$	∞	∞	0	$-\infty$	$-\infty$
$a<0$	∞	classical			$-\infty$
0	0	computing with			0
$a>0$	$-\infty$	finite values			∞
∞	$-\infty$	$-\infty$	0	∞	∞

Table 5. Multiplying infinite values

Division

If we use definition 2.1 for interval division, we get the following expression

$$X/Y = \{x/y : x \in X \wedge y \in Y\}$$

for the case $0 \notin Y$. As mentioned before, in this case the operation is very simple, because we just multiply with the reciprocal of Y. We can reformulate this by eliminating the division:

$$X/Y = \{z : z \cdot y = x \wedge x \in X \wedge y \in Y\}. \tag{3.1}$$

It is obvious that if $0 \notin Y$, this expression is equivalent to the previous one. There are two trivial cases. The first case is, when $0 \in X$. Then from the reformulation follows that the result is the whole real line (because for every x, $x \cdot 0$ is 0). The second case is, when $0 \notin X$ and $Y = [0,0]$. In this case there is no a nonzero number for which $0 \cdot x = a$, so the result is the empty set. The other cases are similar to the cases of multiplication: we need to examine how each interval is situated with respect to zero. The details are in table 6. In this table the angled brackets denote arbitrary interval endpoints. We must clarify the cases when the table contains infinite values in its first column. The rules are simple: if we divide zero or an infinite value with an infinite value, the result is undefined, and is zero otherwise.

Case	Result set
$\overline{X} < 0$ and $\underline{Y} < \overline{Y} = 0$	$\langle \overline{X}/\underline{Y}, \infty)$
$\overline{X} < 0$ and $\underline{Y} < 0 < \overline{Y}$	$(-\infty, \overline{X}/\overline{Y}\rangle \cup \langle \overline{X}/\underline{Y}, \infty)$
$\overline{X} < 0$ and $0 = \underline{Y} < \overline{Y}$	$\langle -\infty, \overline{X}/\overline{Y}\rangle$
$\underline{X} > 0$ and $\underline{Y} < \overline{Y} = 0$	$(-\infty, \underline{X}/\underline{Y}\rangle$
$\underline{X} > 0$ and $\underline{Y} < 0 < \overline{Y}$	$(-\infty, \underline{X}/\underline{Y}\rangle \cup \langle \underline{X}/\overline{Y}, \infty)$
$\underline{X} > 0$ and $0 = \underline{Y} < \overline{Y}$	$\langle \underline{X}/\overline{Y}, \infty)$

Table 6. Cases of interval division

3.1. About the topology

We note first, that the definitions used in the previous section are independent from the type of intervals. This means that we can use them without changing such concepts as united extension, natural extension, interval extension, inclusion isotonic function, rational interval function. The fundamental theorem remains true as well, because its proof depends only on set theoretical considerations. We can use the Moore–distance, but the space is only a quasimetric space. The Lipschitz property is more interesting, but for rational interval functions it is straightforward.

4. Realization

In [4] a relatively simple solution for handling arbitrary intervals has been implemented. There we used the *RealRange*() construction of Maple. Because it is essentially undocumented, this new realization does not use this. Instead we used the module technique of Maple, which resembles the object oriented methods of modern programming environments. The structure of intervals is defined by the following code:

Listing 1. The Interval Object

```
Interval := proc (a, b, lO := true, rO := true)
  module ()
    export Left, Right, LeftOpen, RightOpen, Radius,
        Width,
            MidPoint, Magnitude, Closed, Open,
                ModulePrint, ModuleApply, isInterval;
```

```
    if is(b < a) then
      Left := b; Right := a
    else
      Left := a; Right := b
    end if;
    LeftOpen := lO; RightOpen := rO;
    Width := abs(Left-Right);
    Radius := (1/2)*Width;
    MidPoint := (1/2)*Left+(1/2)*Right;
    Magnitude := max(abs(Left), abs(Right));
    Open := LeftOpen and RightOpen;
    Closed := `not`(LeftOpen or RightOpen);
    isInterval:=true;
    ModulePrint := proc ()
      local s;
      s := "";
      if LeftOpen then s := cat(s, "(")
      else s := cat(s, "[")
      end if;
      s := cat(s, convert(Left, string), ",", convert(
          Right, string));
      if RightOpen then s := cat(s, ")")
      else s := cat(s, "]")
      end if;
    end proc ;
    ModuleApply := proc()
      thismodule:-ModulePrint()
    end proc
  end module
end proc:
```

Here we defined functions usually used in interval arithmetics mentioned earlier. The *ModulePrint* and *ModuleApply* functions realize Maple's familiar echoing facility: it displays the interval in the usual form. The *isInterval* variable indicates for other procedures that it is an interval object.
The interval type is defined as follows:

Listing 2. The Interval Type

```
isInterval := proc (x::anything)
  try
    x:-isInterval
  catch:
    false
```

```
  end try
end proc

TypeTools:-AddType(Interval, IsInterval)
```

And let us see a longer example:

Listing 3. The Interval Multiplication

```
`*`:= overload([
    proc(A::NotInterval,B::Interval)
      option overload;
      if is(A > 0) then
        Interval(B:-Left*A, B:-Right*A, B:-LeftOpen, B
            :-RightOpen)
      else
        Interval(B:-Right*A, B:-Left*A, B:-RightOpen,
            B:-LeftOpen)
      end if
    end proc,
    proc(A::Interval, B::`**`)
      option overload;
      A * Reciprocial(op(1,B))
    end proc,
    proc(A::Interval,B::NotInterval)
      option overload;
      if is(B > 0) then
        Interval(A:-Left*B, A:-Right*B, A:-LeftOpen, A
            :-RightOpen)
      else
        Interval(A:-Right*B, A:-Left*B, A:-RightOpen,
            A:-LeftOpen)
      end if
    end proc,
    proc (A::Interval, B::Interval)
      local a1, a2, b1, b2, c1, c2, la, ra, lb, rb, lc
          , rc, a01, b01, x, y;
      a1 := A:-Left; a2 := A:-Right; la := A:-LeftOpen
          ; ra := A:-RightOpen;
      b1 := B:-Left; b2 := B:-Right; lb := B:-LeftOpen
          ; rb := B:-RightOpen;
      if is(0 <= a1) and is(0 <= b1) then
        c1 := a1*b1; c2 := a2*b2;
        lc := la or lb; rc := ra or rb
```

```
elif is(a1 < 0) and is(0 < a2) and is(0 <= b1)
   then
  c1 := a1*b2; c2 := a2*b2;
  lc := la or rb; rc := ra or rb
elif is(a2 <= 0) and is(0 <= b1) then
  c1 := a1*b2; c2 := a2*b1;
  lc := la or rb; rc := ra or lb
elif is(0 <= a1) and is(b1 < 0) and is(0 < b2)
   then
  c1 := a2*b1; c2 := a2*b2;
  lc := ra or lb; rc := ra or rb
elif is(0 <= a2) and is(b1 < 0) and is(0 < b2)
   then
  c1 := a1*b2; c2 := a1*b1;
  lc := la or rb; rc := la or lb
elif is(0 <= a1) and is(b2 <= 0) then
  c1 := a2*b1; c2 := a1*b2;
  lc := ra or lb; rc := la or rb
elif is(a1 < 0) and is(0 < a2) and is(b2 <= 0)
   then
  c1 := a2*b1; c2 := a1*b1;
  lc := ra or lb; rc := la or lb
elif is(a2 <= 0) and is(b2 <= 0) then
  c1 := a2*b2; c2 := a1*b1;
  lc := ra or rb; rc := la or rb
else #0<=a2<a01 and 0<=b01<b2
  a01 := -a1; b01 := -b1;
  if is(0 <= a01) and is(a01 <= a2) and is(0 <=
     b01) and is(b01 <= b2) then
    c2 := a2*b2; rc := ra or rb; x := a1*b2; y
       := a2*b1;
    if is(x <= y) then c1 := x; lc := la or rb
    else c1 := y; lc := ra or lb
    end if
  elif is(0 <= a2) and is(a2 <= a01) and is(0 <=
     b2) and is(b2 <= b01) then
    c2 := a1*b1; rc := la or lb; x := a1*b2; y
       := a2*b1;
    if is(x <= y) then c1 := x; lc := la or rb
    else c1 := y; lc := ra or lb
    end if
```

```
        elif is(0 <= a01) and is(a01 <= a2) and is(0
           <= b2) and is(b2 <= b01) then
          c1 := a2*b1; lc := ra or lb; x := a1*b1; y
             := a2*b2;
          if is(y <= x) then c2 := x; rc := la or lb
          else c2 := y; ra := ra or rb
          end if
        else
          c1 := a1*b2; lc := la or rb; x := a1*b1; y
             := a2*b2;
          if is(y <= x) then c2 := x; rc := la or lb
          else c2 := y; ra := ra or rb
          end if ;
        end if ;
      end if;
      Interval(c1, c2, lc, rc)
    end proc
  ]);
```

This process overwrites the standard multiplication, so we can use the mathematical symbols for intervals too. In this code, the case, when we multiply an interval with a real number was handled as well. The full source code in Maple and Sage, and examples can be found at the the author's homepage: `https://compalg.inf.elte.hu/~czirbusz/`.

4.1. Applications

We mentioned above that the interval extension is not unique. We can compute the expression $X \cdot (1 - X)$ in two ways. In Maple:

Listing 4. The Interval Function $X \cdot (1 - X)$

```
with(IntervalOperators);
     [`*`, `+`, `-`, `/`, `<`, `>`, Hull, Intersect,
        Reciprocial, Union, `^`]
X := Interval(0,1,false,false);
     X := "[0,1]"
X·(1 − X);
     "[0,1]"
X − X^2;
     "[−1,1]"
unwith(IntervalOperators);
```

The interval operators are defined in the package *IntervalOperators*, so we can use it with the **with** function (and the function **unwith** removes it from the memory). We overloaded the standard arithmetic operators, so we can use the standard addition, subtraction, etc. symbols. The

$$X := Interval(0, 1, false, false)$$

statement declares the closed unit interval.

Examples for basic operations

We illustrate the basic operators in the following listing:

Listing 5. Using the Basic Operators

$A := Interval(1, 2, false, false); B := Interval(5, 4);$
A := "[1,2]"
B := "(4,5)"
$A \cdot B;$
"(4,10)"
$\frac{A}{2};$
"[1/2,1]"
$\frac{1}{B};$
"(1/5,1/2)"
$\frac{A}{B};$
"(1/5,1/2)"

4.2. A symbolic example

This example illustrates the advantages of symbolic computation, as a part of examining regularity of the functional equation below

$$\begin{aligned}&(f(t(x+y)) - f(tx))(f(x+y) - f(y))\\ &= (f(t(x+y)) - f(ty))(f(x+y) - f(x)) .\end{aligned}$$

We try to construct some special compact subset of the interval $[a, b]$, where a is an arbitrary positive real constant, and b will be later determined. The most important step is to create subdivisions of this interval, this can be done in essence automatically, see the code.

Listing 6. A Symbolic Example

```
assume(a :: realcons); additionally(a > 0);
assume(b :: realcons); additionally(b > a);
assume(t :: realcons); additionally(t > 0);
assume(y :: realcons); additionally(y > 0);
assume(t1 :: realcons); additionally(t1 > 0);
assume(y1 :: realcons); additionally(y1 > 0);
assume(x :: realcons); additionally(a <= x and x <= b);

dmn := Interval(a, b);
      dmn := "(a~,b~)"
Divs := Divisions(dmn, 4);
      Divs := ["(a~,3/4*a+1/4*b)", "(3/4*a+1/4*b,1/2*a
          +1/2*b)", "(1/2*a+1/2*b,1/4*a+3/4*b)", "(1/4*
          a+3/4*b,b~)"]
with(IntervalOperators);
Evaluate(g7, x, Divs[1]);
      "((a+y)*t1,(3/4*a+1/4*b+y)*t1)"
```

In this code the function *Divisions* computes the subintervals, the function *Evaluate* replaces the variable in function $g7$ with the subinterval, where $g7$ is the function $t1 * (x + y)$ computed previously.

The computer environment

First of all, we note that Maple follows the standard IEEE 754, so we redefined the multiplication. Various computers were used during development of the program, mainly a Fujitsu Siemens Amilo PI2530 notebook, with 2x Intel(R) Core(TM)2 Duo CPU 1.50GHz with multiple versions of 64 bit Ubuntu Linux operating systems and various Linux kernels, 2059716 kB of total memory, and 71892 kB of swap. The question of speed is not significant, on this machine with Maple©v15 the required time was only $0.01 - 0.03$ seconds. For information about the mentioned computer algebra systems, see their home pages, ie. Maple at [11] and Sage at [16].

References

[1] Arbitrary Precision Real Intervals. "http://www.sagemath.org/doc/reference/sage/rings/real_mpfi.html".

[2] MPFI, a multiple precision interval arithmetic library based on MPFR. "http://perso.ens-lyon.fr/nathalie.revol/software.html".
[3] Interval Computations. Interval and Related Software. "http://www.cs.utep.edu/interval-comp/intsoft.html".
[4] **Czirbusz, S.,** Testing regularity of functional equations with computer, *Aequationes mathematicae*.
[5] **Hansen, E. and G.W. Walster,** *Global Optimization Using Interval Analysis*, Marcel Dekker inc., 2004.
[6] **Hickey, T., Q. Ju and M.H. van Emden,** Interval arithmetic: from principles to implementation. *Journal of the ACM*, **48(5)** (2001), 1038–1068.
[7] **Jaulin, L. and G. Chabert,** Resolution of nonlinear interval problems using symbolic interval arithmetic, *Engineering Applications of Artificial Intelligence*, **23(6)** (2010), 1035–1040.
[8] **Jaulin, L., M. Kieffer, O. Didrit, and A. Walter,** *Applied Interval Analysis*, Springer, 2001.
[9] **Krämer, W. and W. Hofschuster,** intpakX – Verified Numerics meets Computer Algebra, http://www2.math.uni-wuppertal.de/~xsc/software/intpakX/.
[10] **Ulrich, W.,** Kulisch and Universität Karlsruhe, Complete Interval Arithmetic and its Implementation on the Computer.
[11] **Maplesoft,** *Mmaple (Version 15.0)*, 2012. http://www.maplesoft.com/.
[12] R. E Moore. *Interval Analysis*. Prentice Hall Inc., Englewood Cliffs, 1966.
[13] **Moore, R.E.,** *Methods and Applications of Interval Analysis*, SIAM, 1979.
[14] **Moore, R.E., R.B. Kearfott and M.J. Cloud,** *Introduction to Interval Analysis*, SIAM, 2009.
[15] **Neumaier, A.,** *Interval Methods for Systems of Equations*, Encyclopedia of Mathematics and its Application, Cambridge University Press, 1990.
[16] **Stein, W.A. et al.,** *Sage Mathematics Software* (*Version 5.5*), The Sage Development Team, 2013. http://www.sagemath.org.

S. Czirbusz
ELTE University
Budapest
Hungary
czirbusz@compalg.inf.elte.hu