Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Logic and Algebraic Programming 64 (2005) 13–39
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Arbitrary precision real arithmetic: design and algorithms

Valérie Ménissier-Morain

LIP6, Université Paris 6, 8 rue du Capitaine Scott, F-75015 Paris, France

Abstract

We describe here a representation of computable real numbers and a set of algorithms for the elementary functions associated to this representation.

A real number is represented as a sequence of finite B-adic numbers and for each classical function (rational, algebraic or transcendental), we describe how to produce a sequence representing the result of the application of this function to its arguments, according to the sequences representing these arguments. For each algorithm we prove that the resulting sequence is a valid representation of the exact real result.

This arithmetic is the first real arithmetic with mathematically proved algorithms.

1. Introduction

1.1. Motivation

We try to determine here what should be an arithmetic for a modern and reliable programming language.

The exactitude of the arithmetic is clearly an essential feature of a reliable programming language, but we show here that even a floating point representation with variable length, as it exists in symbolic computation softwares, is insufficient. Furthermore this arithmetic gives its users some wrong ideas, as disastrous as the round-off errors themselves. The floating point arithmetic [1] is obviously not an exact arithmetic: each partial result is systematically rounded off, and these successive round-off errors may lead to completely erroneous answers for ill-conditioned problems like the computation of $1/(y - x)$ where x is "much greater" than 1 (for example $x = 10^{20}$) and $y = x + 1$. For this particular computation, a single precision floating point computation induces an error due to a division

E-mail address: Valerie.Menissier-Morain@lip6.fr
URL: http://www-calfor.lip6.fr/$\sim$vmm.

1567-8326/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2004.07.003

by zero, but an exact computation yields a result, 1, that is furthermore exactly represented in a floating point arithmetic. However, in order to handle rational or real numbers, one represents them generally with the floating point arithmetic supplied by the computer with a fixed number of significant digits. The programmer is usually aware of these round-off problems, but nevertheless, he remains confident of the computed results because of "intuitive properties" of the floating point arithmetic. Among these pseudo-properties, one can cite:

(1) even if the result is not rigorously exact, it is certainly close to the exact result, that is to say that the round-off errors will be minor and in particular the order of magnitude is supposed to be preserved;
(2) a few floating point operations can only induce a slight inaccuracy in the computed result;
(3) the rounding mode has little effect on the quality of the result;
(4) if a result is computed with more digits, its accuracy will be better. More precisely, we hope that the distance between the computed value and the real value decreases according to the number of digits in the computation;
(5) the result does not depend on the arithmetic, the computer, etc.

There are now numerous examples (see (2–4), etc.) where these common ideas are trampled on:

(1) In [2], Jean-Michel Muller presents a rational sequence whose theoretical limit is 6, and after 10 (resp. 20) iterations for IEEE single (resp. double precision) the value of computed terms is always 100. The convergence to this wrong limit is fast and stable.

    The floating point arithmetic does not even preserve the order of magnitude of the limit. The floating point terms are rapidly irrelevant, even though the computation of these terms leads to very few operations. The speed of the convergence and the stability of the limit ensure neither the accuracy nor the order of magnitude of the computed limit.

    Any round-off error ejects the sequence out of the repulsive basin for 6 and project it in the attractive basin for 100.
(2) In [3], Jean-Michel Muller presents a sequence for which the 25th term is about 0.04, the result is about -10^{14} on a pocket computer and $+4 \times 10^9$ on a workstation.
(3) In [4], Jean-Marie Chesneaux presents a second degree equation with a double root. According to the rounding mode, we obtain the real double root or two real roots or two complex roots.

Some of these examples concern rather single precision than double precision floating point arithmetic and simply increasing the precision of the floating point representation is sufficient to solve the problem, but other examples will probably emerge even when employing higher-precision floating point arithmetic.

1.1.1. Discussion

These examples might be considered as very particular problems, and thus we can consider that it is necessary for these special cases only to analyze the computation and the evolution of the precision without any modification of the arithmetic of the language.

But this analysis is so tedious that in practice the programmer ignores it.

In fact we have no easy way to recognize such problems and, furthermore, some traditional areas of floating point computations such as computer graphics and scientific computation show such problems in their usual practice. For example, in this second domain,

algorithms produce positive definite matrices so a algorithm adapted to such matrices is used, but the floating point computation of the matrix may produce a not positive definite matrix so the result of the second algorithm is doubtful: at best it fails, it may even loop, at worst a completely wrong result is returned with almost no way to control its correctness. In other cases (for example the crash of the first try of Ariane 5), floating point computation fails because of a floating point overflow or underflow that the programmer didn't foresee. These areas are much concerned with money and safety (rockets, airplanes design structures).

Yet the programmer desires reliable arithmetic results, so the correction must be ensured by the programming language that should perform automatically a round-off error analysis to get back a real confidence in results obtained with floating point arithmetic.

1.1.2. Solutions

What solution can we envisage to use?

The simplest way to deal with most floating-point anomalies at the present time is simply to use higher-precision arithmetic, but we cannot know what precision will be necessary to solve the problem.

The most usual solution to this question is interval analysis [5–8]. The computation is performed using floating point arithmetic and propagates during all this computation an upper bound of the round-off error for rational operations, according to the IEEE-754 standard, so that one obtains at the end of the computation a floating point result and an upper bound for the round-off error on this result. Interval arithmetic can be only twice as expensive as ordinary floating point computations, at least in theory. This analysis indicates when the result is wrong. For the preceding sequences, a computation with such an arithmetic will indicate that the upper bound for the round-off error on this result is very big. However if we want to compute the exact result or a result as close as we want of the exact value and not only that the result of the floating point computation is completely wrong, this solution is not satisfactory.

We want to fundamentally avoid the difficulties inherent in any floating-point arithmetic, no matter how precise. We give here an answer to the programmer who needs reliable arithmetic results for which the correction is ensured by the programming language and not by an arithmetic that indicates an upper bound for the distance between the exact value and the obtained result. The programmer indicates an upper bound for the final round-off error and this bound is respected all along the computation, even if it requires a very high precision in a particular step of the computation. In some sense, this analysis is the reciprocal analysis of what would be a complete interval analysis (that is to say that concerns any classical elementary function).

The ML language was designed for safe programming and so should ensure safety in numerical programming. So we implement a small prototype of an arbitrary precision library in this language according to the work described here.

1.2. History of the problem

This work took place in 1994. Other solutions appeared after 1994 and are presented in Section 7 at the end of this article.

Brent in [9] described in 1976 algorithms to compute quickly multiple-precision evaluations of elementary functions, but he did not consider real numbers as full members of

a language. He computes the image of a floating-point number x (thus a rational number) by an algebraic or transcendental function f to a precision $\mathcal{O}(2^{-n})$. This work is however interesting from a practical point of view.

1.2.1. Redundant sequences of digits

Wiedmer proposed in 1977 a solution for real number computations in [10–12]. This solution can be considered as unbounded on-line arithmetic. However, Wiedmer proposed only an algorithm to add real numbers. These ideas were studied again and extended by Boehm in [13,14]. Computable real numbers are represented by an infinite sequence of digits in a given base B. For such a representation the digits of the results are produced "from left to right", beginning with the most significant digits, in opposition to the usual algorithms for addition and multiplication for example, but this technique is common for on-line arithmetic. Particularly, Avizienis in [15] and Wiedmer in [12] proved that an addition algorithm "from left to right" implies the redundancy of the representation: for example, digits are in the integer interval $[-B+1, B-1]$ rather than the classical interval $[0, B-1]$. The idea is that it is necessary to anticipate what will be the next digits of the arguments of the addition algorithm and for example to overestimate by one the absolute value of the sum, even if one needs to correct this trend on the next produced digit by a negative sign. We studied this representation, described and proved algorithms for rational operations, but we did not work out so far algorithms for transcendental functions. Perhaps the Cordic algorithms described by Lin and Sips in [16] may be used to compute these functions. The incrementality is a natural good point for this representation: if one need some more digits, one starts from the list of already computed digits rather than from the beginning of the computation. However, apart from the lack of well-integrated algorithms for transcendental functions, the algorithms for rational operations are intricate and rather inefficient.

1.2.2. Computable Cauchy sequences

In [13,14], Boehm studied a more natural representation. This representation is designed for almost automatic evaluation of round-off errors in programs written in Fortran. In his implementation, the classical operations on floating point numbers are transparently replaced by exact operations on real numbers, then some numerical tests of small size are performed with each arithmetic, so that if a floating point result does not correspond to the expected value, one can attribute this computation error either to a round-off error if the real result is correct or to an error in the implementation of the algorithm by the program if the real result is wrong. Boehm describes algorithms for addition and multiplication on this representation.

Boehm has developed an implementation for each of these two representations. The comparison of the running times indicates clearly that the second one is much faster than the first one.

We have studied this second representation and now we propose a complete and entirely proved set of algorithms for all elementary functions. This work leads to an implementation in the Caml implementation of the ML language.

1.2.3. Continued fractions

Finally, in [17–19], Vuillemin interprets Bill Gosper's work on the continued fractions arithmetic (essentially rational operations) [20] and represents real numbers by continued fractions, with the underlying idea that continued fractions are the "closest" rational numbers to the real numbers. However, apart from the fact that these algorithms are prin-

cipally not proved, this representation is rather inadequate to the architecture of current computers so it is inefficient. The author have implemented a complete prototype for this representation that exhibits poor running times despite the natural incrementality of the method.

We present these three representations with all details in [21]. We describe in this article the second representation mentioned above.

1.3. Summary

We first recall the main properties of computable real numbers. We deduce from one definition, among the three definitions of this notion, a representation of these numbers as sequence of finite B-adic numbers and then we describe algorithms for rational operations and transcendental functions for this representation. Then we describe briefly the prototype written in Caml. Finally we present briefly the evolution of the domain since 1994.

2. Computable real numbers

We present here a short summary of the properties of computable real numbers. No proofs will be provided in this section. For more details see [22].

2.1. Definitions

There are several definitions for computable real numbers, we will use here only the definition as B-approximable real number since it is the nearest of our work (see [22] for other definitions and equivalences).

We will define the related notion of *finite B-adic numbers* for a given base B and deduce the notion of *B-approximable real number*.

Definition 1 (*Finite B-adic number*). If B is an integer greater than or equal to 2, a rational number r is called a *finite B-adic number* if there exists two integers p and q such that $r = p/B^q$ and q is a positive integer.

This definition generalizes the notion of *dyadic* number. We define now the notion of *B-approximable real number*.

Definition 2 (*B-approximable real number*). A real number x is called B-approximable if there exists a recursive function g such that, for any integer N, $g(N)$ is a finite B-adic number and

$$|x - g(N)| < \frac{1}{B^N}.$$

2.2. Properties of $\mathcal{R}$

We denote up to the end of this section by $\mathcal{R}$ (resp. $\mathcal{C}$) the set of computable real (resp. complex) numbers and as usual by $\mathbb{R}$ (resp. $\mathbb{C}$) the set of real (resp. complex) numbers. The set $\mathcal{R}$ has the following properties:

Theorem 3

(1) *The set $\mathcal{R}$ with the addition and the multiplication of $\mathbb{R}$ is an Archimedean commutative field.*
(2) *The set $\mathcal{C}$ with the addition and the multiplication of $\mathbb{C}$ is an algebraically closed commutative field.*

Theorem 4

(1) *$\mathcal{C}$ is closed for elementary functions.*
(2) *$\mathcal{R}$ is closed for* exp, log, $(x, y) \mapsto x^y$, sin, cos, tan, sinh, cosh, tanh, arcsin, arccos, arctan, arcsinh, arccosh, arctanh.

Theorem 5

(1) *$\mathcal{R}$ is a denumerable subset of $\mathbb{R}$ and is dense in $\mathbb{R}$.*
(2) *$\mathcal{C}$ is a denumerable subset of $\mathbb{C}$.*

2.3. Undecidability theorems about $\mathcal{R}$

Theorem 6

(1) *There exists no general algorithm to determine whether a computable real number is zero or not.*
(2) *There exists no general algorithm to determine the image of a computable real number by a function with a discontinuity at this point.*
(3) *There exists no general algorithm to determine if a computable real number is greater than another one.*
(4) *There exists no general algorithm to determine the integer part of a computable real number.*
(5) *There exists no general algorithm to determine if a computable real number is rational.*

A consequence of the fourth proposition of this theorem is that one cannot determine exactly the classical continued fraction expansion or the development in a given base of any computable real number.

However, one should not attach an excessive importance to these impossibilities because according to the last definition of computable real numbers, any computable real numbers may be known to a precision within B^{-n} in a given base B for any integer n. As far as the comparison is concerned, the following theorem establishes that if two computable real numbers differ, then there is an algorithm that indicate which one is the greater one and at which rank their definition sequences differ.

Theorem 7. *Let $A = (a_n)_{n\in\mathbb{N}}$ and $B = (b_n)_{n\in\mathbb{N}}$ be two recursive Cauchy sequences of rational numbers with distinct respective limits a and b, then there exists an algorithm to compare a and b that terminates.*

3. Description of a representation of computable real numbers with particular sequences of B-adic numbers

Computable real numbers will be represented here by B-adic numbers and as in Boehm's work, we represent the B-adic numbers by longer and longer integer corresponding to the numerator of B-adic approximations more and more precise.

It is well-known that the limit of a sequence of B-adic numbers $(a_n/B^{k_n})_{n\in\mathbb{N}}$ is also the limit of this other sequence of B-adic numbers $(b_n/B^n)_{n\in\mathbb{N}}$ with $b_n = \lfloor a_n B^{n-k_n} \rfloor$. This second sequence is interesting because the denominator of each B-adic is exactly B raised to its rank in the sequence so we need only the sequence of integers $(b_n)_{n\in\mathbb{N}}$ to represent the limit of this sequence.

Thus we approximate a computable real number r with a sequence of integers $(c_n)_{n\in\mathbb{N}}$ such that $|r - c_n B^{-n}| < B^{-n}$ for any integer n.

We present now precisely the definitions and general properties of this representation to prepare the algorithms for elementary functions for this representation. Proofs are rather short and simple, we give here only the longest proof (for Property 15). All details are available in [22].

Let B be a given base, i.e. an integer greater than or equal to 2. A computable real number is represented by a sequence of integers that satisfy the following property:

Definition 8 (*Bounds property*). Let x be a computable real number, for any integer p, the *bounds property* of x by p for order n is characterized by the following inequality $\left|x - pB^{-n}\right| < B^{-n}$ i.e. $(p-1)B^{-n} < x < (p+1)B^{-n}$.

We authorize negative indices for practical reasons because sometimes we need only to know the order of magnitude of a real number rather than its integer part. The bounds property apply easily to negative indices and it saves some time during the computation. We will now express some properties of the integers that satisfy the bounds property for a given real number and a given order.

Property 9. *Let x be a computable real number, n an integer and p be an integer. Suppose that the bounds property of x by p for order n is satisfied. Then $p = \lfloor B^n x \rfloor$ or $p = -\lfloor B^n(-x) \rfloor$. Furthermore if $B^n x$ is an integer then $p = \lfloor B^n x \rfloor$.*

The real numbers that we will consider in this section are computable real numbers x represented by sequences of integers $(x_n)_{n\in\mathbb{N}}$ such that the bounds property for x by x_n for order n is satisfied.

We will now define the sign function before we express the following property that describes the relations between the integers that satisfy the bounds property for x and $|x|$ for the same order.

Definition 10 (*sign*). The function sign is defined from $\mathbb{R}$ to $\{-1, 0, 1\}$ with the usual following equality:

$$\operatorname{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$

Practically, for each non-zero real number x and for each integer n, the sign of x is the sign of each non-zero value x_n since if $x_n > 0$, then $x > (x_n - 1)B^{-n} \geqslant (1-1)B^{-n} = 0$ and if $x_n < 0$, then $x < (x_n + 1)B^{-n} \leqslant (-1+1)B^{-n} = 0$. Furthermore its computation terminates for any not null number.

Property 11. *Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$ and n be an integer, $|x_n|$ satisfies the bounds property of $|x|$ for order n and if p is a positive*

integer that satisfies the bounds property of $|x|$ for order n, then the integer q defined by $q = \text{sign}(x) \times p$ satisfies the bounds property of x for order n.

We have also the following technical properties:

Property 12. *Let x, α be two real numbers and n be an integer. If $\alpha B^{-n} < x < (\alpha + 1)B^{-n}$, then $\lfloor \alpha \rfloor + 1$ and $\lceil \alpha \rceil$ satisfy the bounds property of x for order n.*

Property 13. *Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, n and m be integers such that $n \leqslant m$, then the integer $\left\lfloor \frac{x_m}{B^{m-n}} \right\rfloor$ satisfies the bounds property of x for order n.*

For efficiency reasons, the implemented representation includes for each real number x represented by a sequence $(x_n)_{n\in\mathbb{Z}}$ the most precise approximation ever computed for x, $x_{\text{mpa}(x)}$ for order $\text{mpa}(x)$. In this way, any approximation less precise for x may be computed by a simple shift operation on $x_{\text{mpa}(x)}$ rather than by a possibly complex computation that we have in some sense already performed before:

$$\text{if } n \leqslant \text{mpa}(x), \text{ then we take } x_n = \left\lfloor \frac{x_{\text{mpa}(x)}}{B^{\text{mpa}(x)-n}} \right\rfloor.$$

The value of x_n may slightly vary according to the $\text{mpa}(x)$ value.

We will now define the msd function that indicates the order of magnitude of a real number.

Definition 14 (*msd*). The function msd ("most significant digit") is defined from $\mathbb{R}$ to $\mathbb{Z}$ for any real number x represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, by the equality $\text{msd}(x) = \min_{n\in\mathbb{Z}}(|x_n| > 1)$.

Practically, the function msd is recursively computed and does not terminate for zero. This function satisfies the following properties:

Properties 15

(1) *For any non-zero real number x,* $\text{msd}(x)$ *exists and is unique* (*at the exact time of its computation, see the remark below*), *with* $2 \leqslant \left|x_{\text{msd}(x)}\right| \leqslant 2B$ *and* $\text{msd}(x) = -\left\lfloor \log_B |x| \right\rfloor$ *or* $\text{msd}(x) = -\left\lfloor \log_B |x| \right\rfloor + 1$.

(2) *For any non-zero real number x and any integer $n < \text{msd}(x)$, we have $|x_n| \leqslant 1$.*

(3) *For any non-zero real number x and any integer $n \geqslant \text{msd}(x)$, then*

$$B^{n-\text{msd}(x)} \leqslant |x_n| \leqslant B^{n-\text{msd}(x)}(2B + 1)$$

and

$$1 \leqslant \left| \left\lfloor \frac{x_n}{B^{n-\text{msd}(x)}} \right\rfloor \right| \leqslant 2B + 1.$$

Proof. Let x be a not null real number, we will prove there exists an integer n such that $\left|x_n\right| > 1$.

By definition of $x \mapsto \lfloor x \rfloor$, we have

$$B^{\lfloor \log_B |x| \rfloor} \leqslant |x| < B^{\lfloor \log_B |x| \rfloor + 1} \tag{1}$$

but according to property (11), $\left|x_{-\lfloor \log_B |x| \rfloor}\right|$ satisfies the bounds property for order $-\left\lfloor \log_B |x| \right\rfloor$ of $|x|$:

$$\frac{\left|x_{-\lfloor \log_B |x| \rfloor}\right| - 1}{B^{-\lfloor \log_B |x| \rfloor}} < |x| < \frac{\left|x_{-\lfloor \log_B |x| \rfloor}\right| + 1}{B^{-\lfloor \log_B |x| \rfloor}}.$$

We combine these two inequalities and we obtain

$$\frac{\left|x_{-\lfloor \log_B |x| \rfloor}\right| - 1}{B^{-\lfloor \log_B |x| \rfloor}} < B^{\lfloor \log_B |x| \rfloor + 1}$$

and

$$B^{\lfloor \log_B |x| \rfloor} < \frac{\left|x_{-\lfloor \log_B |x| \rfloor}\right| + 1}{B^{-\lfloor \log_B |x| \rfloor}}.$$

We reduce to lowest terms and obtain the following inequalities between the concerned numerators: $\left|x_{-\lfloor \log_B |x| \rfloor}\right| - 1 < B$ and $1 < \left|x_{-\lfloor \log_B |x| \rfloor}\right| + 1$.

These inequalities concern integers thus we deduce that $\left|x_{-\lfloor \log_B |x| \rfloor}\right| \leqslant B$ and $1 \leqslant \left|x_{-\lfloor \log_B |x| \rfloor}\right|$ that is to say $1 \leqslant \left|x_{-\lfloor \log_B |x| \rfloor}\right| \leqslant B$.

Let us suppose that $\left|x_{-\lfloor \log_B |x| \rfloor}\right| = 1$ and prove that $1 < \left|x_{-\lfloor \log_B |x| \rfloor + 1}\right|$.

According to the bounds property of $|x|$ for order $-\left\lfloor \log_B |x| \right\rfloor + 1$ satisfied by $\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right|$, we have

$$\frac{\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| - 1}{B^{-\lfloor \log_B |x| \rfloor + 1}} < |x| < \frac{\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| + 1}{B^{-\lfloor \log_B |x| \rfloor + 1}}.$$

We combine this inequality with (1), we obtain

$$\frac{\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| - 1}{B^{-\lfloor \log_B |x| \rfloor + 1}} < B^{\lfloor \log_B |x| \rfloor + 1}$$

and

$$B^{\lfloor \log_B |x| \rfloor} < \frac{\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| + 1}{B^{-\lfloor \log_B |x| \rfloor + 1}}.$$

We reduce to lowest term and obtain the following inequalities between the concerned numerators: $\left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| - 1 < B^2$ and $B < \left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| + 1$, that is to say $B \leqslant \left|x_{-\lfloor \log_B |x| \rfloor + 1}\right| \leqslant B^2$ and a fortiori $1 < \left|x_{-\lfloor \log_B |x| \rfloor + 1}\right|$ since $B \geqslant 2$.

Since we consider the smallest n such that $|x_n| > 1$, the result is completely determined when the computation is performed even if it may vary according to the computed approximation of x.

We will now prove that for any $n < -\left\lfloor \log_B |x| \right\rfloor$, we have $|x_n| \leqslant 1$. According to the bounds property of $|x|$ satisfied by $|x_n|$ for order n, we have

$$\frac{|x_n| - 1}{B^n} < |x|$$

and according to (1), we have $|x| < B^{\lfloor \log_B |x| \rfloor + 1}$. We deduce from these two inequalities that $|x_n| - 1 < B^{n + \lfloor \log_B |x| \rfloor + 1}$. But, according to an hypothesis, we have $n + \left\lfloor \log_B |x| \right\rfloor + 1 \leqslant 0$, thus $|x_n| - 1 < 1$ and since this inequality concerns integers, we have $|x_n| \leqslant 1$.

We will now prove the inequalities for $\left|x_{\mathrm{msd}(x)}\right|$. If $\mathrm{msd}(x) = -\left\lfloor \log_B |x| \right\rfloor$, then we have $1 < \left|x_{\mathrm{msd}(x)}\right| \leqslant B$.

If $\mathrm{msd}(x) = -\left\lfloor \log_B |x| \right\rfloor + 1$, we have $\left|x_{-\lfloor \log_B |x| \rfloor}\right| = 1$, thus $\left|x_{-\lfloor \log_B |x| \rfloor}\right| + 1 = 2$ and according to the bounds property of $|x|$ for order $-\left\lfloor \log_B |x| \right\rfloor$ satisfied by $\left|x_{-\lfloor \log_B |x| \rfloor}\right|$, $|x| < 2B^{\lfloor \log_B |x| \rfloor}$ and $\left|x_{\mathrm{msd}(x)}\right| - 1 < 2B$, thus $2 \leqslant \left|x_{\mathrm{msd}(x)}\right| \leqslant 2B$.

The lower bound could be reach in an obvious way. The upper bound $|x_{\mathrm{msd}(x)}| \leqslant 2B$, may also be reach as illustrated on the following example: let us choose $x = 2 - 1/(2B)$, $x_0 = 1$, $x_1 = 2B$ with $\mathrm{msd}(x) = 1$.

Let n be an integer such that $n \geqslant \mathrm{msd}(x)$.

According to the definition of $x_{\mathrm{msd}(x)}$, we have

$$\left(\left|x_{\mathrm{msd}(x)}\right| - 1\right) B^{n-\mathrm{msd}(x)} B^{-n} < |x| < \left(\left|x_{\mathrm{msd}(x)}\right| + 1\right) B^{n-\mathrm{msd}(x)} B^{-n},$$

thus

$$\left(\left|x_{\mathrm{msd}(x)}\right| - 1\right) B^{n-\mathrm{msd}(x)} \leqslant B^n x \leqslant \left(\left|x_{\mathrm{msd}(x)}\right| + 1\right) B^{n-\mathrm{msd}(x)}$$

and

$$\left(\left|x_{\mathrm{msd}(x)}\right| - 1\right) B^{n-\mathrm{msd}(x)} \leqslant |x_n| \leqslant \left(\left|x_{\mathrm{msd}(x)}\right| + 1\right) B^{n-\mathrm{msd}(x)}.$$

But $2 \leqslant \left|x_{\mathrm{msd}(x)}\right| \leqslant 2B$, thus $B^{n-\mathrm{msd}(x)} \leqslant |x_n| \leqslant B^{n-\mathrm{msd}(x)}(2B+1)$.

We suppose that $|x_n| \geqslant 2B\, B^{n-\mathrm{msd}(x)} + 1$. We already know that this is impossible if $n = \mathrm{msd}(x)$ so we can consider directly that $n - \mathrm{msd}(x) \geqslant 1$. Thus we have $B^{\mathrm{msd}(x)-1}|x| > (|x_n| - 1)B^{-(n-\mathrm{msd}(x)+1)} \geqslant 2$ thus $|x_{\mathrm{msd}(x)-1}| \geqslant 2$ that is refuted by the minimality of $\mathrm{msd}(x)$. Consequently

$$\frac{|x_n|}{B^{n-\mathrm{msd}(x)}} \leqslant 2B$$

and

$$\left\lfloor \frac{|x_n|}{B^{n-\mathrm{msd}(x)}} \right\rfloor \leqslant 2B.$$

But we may have $B^{n-\mathrm{msd}(x)} \leqslant |x_n| < B^{n-\mathrm{msd}(x)} + 1$ for $n \geqslant \mathrm{msd}(x) + 1$ as illustrated in the following example: let us choose $x = 1 + 1/2B$, $x_0 = 2$, $x_1 = B$, $\mathrm{msd}(x) = 0$ and $n = 1$. □

Let us notice that the value of x_n and of $msd(x)$ may vary of one unit according to the value to $\mathrm{mpa}(x)$. However the rather strict definition that we give of this function ensures, independently of the value of $\mathrm{mpa}(x)$, the fundamental property we wanted to ensure, i.e. $1 < \left|x_{\mathrm{msd}(x)}\right| \leqslant B$ whatever the value of $\mathrm{mpa}(x)$ is when $\mathrm{msd}(x)$ is computed.

We present now a complete set of algorithms to compute elementary functions for this representation, using the corresponding algorithms for rational numbers.

4. Algorithms for the usual elementary functions

4.1. Introduction to algorithms for the computation of elementary functions on $\mathcal{R}$

We will now describe algorithms for computing elementary functions on $\mathcal{R}$. We associate to each elementary function $f : \mathbb{R}^p \to \mathbb{R}$ its representation $\overline{f} : \mathcal{R}^p \to \mathcal{R}$ and for any computable real x, we note $\overline{x}$ its representation.

For each elementary function f with p arguments $(x_1, \ldots, x_p)$, for any integer n and for any $1 \leqslant i \leqslant p$, we have to describe to what precision k_i each argument x_i is supposed

to be computed in a $\overline{x_i}_{k_i}$ approximation and give a formula to apply to these approximations to produce $\overline{f}(\overline{x_1}, \ldots, \overline{x_p})_n$.

After this description of the algorithm we have to establish a theorem of correction as follows:

Theorem (*Correctness of the algorithm for computing a function f on computable real arguments* $(x_1, \ldots, x_p)$*):*
The sequence $\overline{f}(\overline{x_1}, \ldots, \overline{x_p})$ is a representation of $f(x_1, \ldots, x_p)$.

In other words, for any order n, $\overline{f}(\overline{x_1}, \ldots, \overline{x_p})_n$ satisfies the bounds property of $f(x_1, \ldots, x_p)$. This can be also interpreted as the fact that it is right to define $\overline{f(x_1, \ldots, x_p)}$ as $\overline{f}(\overline{x_1}, \ldots, \overline{x_p})$.

These algorithms are designed for any integer $B \geqslant 2$. In almost all algorithms we distinguish the case where $B \geqslant 4$ and $B = 2$ or 3. In fact, for $B \geqslant 4$ we can generally give more precise constants and to distinguish this case rather than to adopt the constants determined by the formulas for $B \geqslant 2$. This should improve the efficiency of the algorithms. We could of course distinguish other cases for which the constant may be still smaller but we have to stop this sequence of improvements and in fact the distinction of $B \geqslant 4$ is relevant according to the implementation (see Section 5).

Theorems and proofs for the correctness of algorithms are very long and tedious, consequently we present here only the proof for multiplication as a representative algorithm and the other theorems and proofs are available in [22].

4.2. Algorithms for rational operations

We will now describe the representation of the image of any computable real number by an elementary function and we begin with the heart of these algorithms: the representation of rational numbers.

Algorithm 1 (*Representation of rational numbers*). Each rational number q is represented by the sequence of integers $(q_n)_{n\in\mathbb{N}}$ such that q_n is defined, for any integer n by the equality:

$$q_n = \left\lfloor B^n q \right\rfloor.$$

Algorithm 2 (*Addition of real numbers*). Let x and y be two real numbers represented by the sequences $(x_n)_{n\in\mathbb{Z}}$ and $(y_n)_{n\in\mathbb{Z}}$ respectively, we represent the sum of these two numbers $x + y$ by the sequence $(\overline{x+y}_n)_{n\in\mathbb{Z}}$ such that:

$$\overline{x+y}_n = \left\lfloor \frac{x_{n+w} + y_{n+w}}{B^w} \right\rceil \quad \text{with } w = \begin{cases} 1 & \text{if } B \geqslant 4 \\ 2 & \text{if } B = 2 \text{ or } 3. \end{cases}$$

Algorithm 3 (*Opposite of a real number*). Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, we represent the opposite of this number $-x$ by the sequence $(\overline{-x}_n)_{n\in\mathbb{Z}}$ such that:

$$\overline{-x}_n = -x_n.$$

Algorithm 4 (*Multiplication of two real numbers*). Let x and y be two real numbers represented by the sequences $(x_n)_{n\in\mathbb{Z}}$ and $(y_n)_{n\in\mathbb{Z}}$ respectively, we represent the product of these two numbers $x + y$ by the sequence $(\overline{x\times y}_n)_{n\in\mathbb{Z}}$ such that:

$$\overline{x\times y}_n = \mathrm{sign}(x_{p_x}) \times \mathrm{sign}(y_{p_y}) \times \left\lceil \frac{1 + |x_{p_x} \times y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil$$

with $p_x = \max(n - \mathrm{msd}(y) + v, \lfloor (n+w)/2 \rfloor)$

and $p_y = \max(n - \mathrm{msd}(x) + v, \lfloor (n+w)/2 \rfloor)$

and $(v, w) = \begin{cases} (3,2) & \text{if } B \geqslant 4 \\ (3,3) & \text{if } B = 3 \\ (4,3) & \text{if } B = 2 \end{cases}$

Remark. The computation of $\mathrm{msd}(x)$ is restricted here by the evaluation of the maximum in expression p_y, that is to say that for any integer k from 0 (beginning of the recursion in the computation of $\mathrm{msd}(x)$) to $n + v - \lfloor (n+w)/2 \rfloor$ (maximum value of $\mathrm{msd}(x)$ for which p_y is determined by the first term in the maximum expression) $x_k = 0$, then p_y is determined by the second term and we stop the computation of $\mathrm{msd}(x)$ and in this way multiplication by 0 terminates. This analysis is of course identical for the evaluation of $\mathrm{msd}(y)$ inside the computation of p_x.

Theorem 16. *For any integer n, we have*

$$(\overline{x\times y}_n - 1)B^{-n} < x \times y < (\overline{x\times y}_n + 1)B^{-n}.$$

If the computation of x and y terminates, then the computation of $x \times y$ terminates too.

Proof (*Correctness of the multiplication algorithm*). First of all, we remark that $p_x + p_y - n \geqslant 2 \times (\lfloor (n+w)/2 \rfloor) - n \geqslant w - 1$.

If $|x_{p_x}| = 0$ and $|y_{p_y}| = 0$, then we have

$$B^n |x \times y| < \frac{(|x_{p_x}| + 1)\,(|y_{p_y}| + 1)}{B^{p_x+p_y-n}} \leqslant \frac{1}{B^{p_x+p_y-n}} \leqslant \frac{1}{B^{w-1}}$$

since $p_x + p_y - n \geqslant w - 1$. According to the definition of w, we have $\frac{1}{B^{w-1}} \leqslant 1$ and $-\frac{1}{B^n} < x \times y < \frac{1}{B^n}$. Consequently $\overline{x\times y}_n = 0$ satisfies the bounds property of $x \times y$ for order n.

We will now consider the last case, that is to say, if at least one of the absolute values $|x_{p_x}|$ and $|y_{p_y}|$ is greater or equal to 1. Because of the symmetry of the problem, we can decide, to reduce the combinatorics of this case by case analysis, that x is concerned.

We have:

$$\frac{1 + |x_{p_x}\, y_{p_y}| - (|x_{p_x}| + |y_{p_y}|)}{B^{p_x+p_y-n}} < B^n |x \times y| < \frac{1 + |x_{p_x}\, y_{p_y}| + (|x_{p_x}| + |y_{p_y}|)}{B^{p_x+p_y-n}}.$$

We will prove that from

$$\frac{|x_{p_x}| + |y_{p_y}|}{B^{p_x+p_y-n}} \leqslant \frac{1}{2},$$

we can deduce the correctness of the algorithm. Let us suppose that this property is satisfied, then we have

$$\frac{1 + |x_{p_x}\, y_{p_y}|}{B^{p_x+p_y-n}} - \frac{1}{2} < B^n |x \times y| < \frac{1 + |x_{p_x}\, y_{p_y}|}{B^{p_x+p_y-n}} + \frac{1}{2}.$$

According to the definition of $x \mapsto \lfloor x \rceil$, we have:

$$\frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} - \frac{1}{2} < \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil \leqslant \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} + \frac{1}{2},$$

thus

$$\left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil - 1 \leqslant \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} - \frac{1}{2} < B^n\,|x \times y|$$
$$< \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} + \frac{1}{2} < \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil + 1$$

that is to say that

$$\left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil$$

satisfies the bounds property of $|x \times y|$ for order n.

According to proposition 11,

$$\mathrm{sign}(x \times y) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil$$

satisfies the bounds property of $x \times y$ for order n.

We have to prove that

$$\mathrm{sign}(x \times y) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil = \mathrm{sign}(x_{p_x}) \times \mathrm{sign}(y_{p_y}) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil$$

to be able to deduce that $\overline{x \times y}_n$ satisfies the bounds property of $x \times y$ for order n.

First of all, it is clear that $\mathrm{sign}(x \times y) = \mathrm{sign}(x) \times \mathrm{sign}(y)$. Since $|x_{p_x}| \geqslant 1$, $\mathrm{sign}(x) = \mathrm{sign}(x_{p_x})$. Thus we have

$$\mathrm{sign}(x \times y) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil = \mathrm{sign}(x_{p_x}) \times \mathrm{sign}(y) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil.$$

If $y_{p_y} = 0$, then

$$\left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil = \left\lfloor \frac{1}{B^{p_x+p_y-n}} \right\rceil = 0 = \overline{x \times y}_n$$

if $|y_{p_y}| > 0$, $\mathrm{sign}(y) = \mathrm{sign}(y_{p_y})$ and

$$\mathrm{sign}(x \times y) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil = \mathrm{sign}(x_{p_x}) \times \mathrm{sign}(y_{p_y}) \times \left\lfloor \frac{1+|x_{p_x}\ y_{p_y}|}{B^{p_x+p_y-n}} \right\rceil = \overline{x \times y}_n.$$

We have now to prove that

$$\frac{|x_{p_x}| + |y_{p_y}|}{B^{p_x+p_y-n}} \leqslant \frac{1}{2}.$$

Since $|x_{p_x}| \geqslant 2$, then $p_x \geqslant \mathrm{msd}(x)$ and $|x_{p_x}| \leqslant 2B\ B^{p_x-\mathrm{msd}(x)}$.

If $p_y \geqslant \mathrm{msd}(y)$, we have in the same way $|y_{p_y}| \leqslant 2B\ B^{p_y-\mathrm{msd}(y)}$ and we have the following inequality:

$$\frac{|x_{p_x}| + |y_{p_y}|}{B^{p_x+p_y-n}} \leqslant 2B \left(\frac{B^{p_x-\mathrm{msd}(x)} + B^{p_y-\mathrm{msd}(y)}}{B^{p_x+p_y-n}} \right) = 2B \left(\frac{B^n}{B^{p_y+\mathrm{msd}(x)}} + \frac{B^n}{B^{p_x+\mathrm{msd}(y)}} \right).$$

But $p_y + \mathrm{msd}(x) \geqslant n + v$ and $p_x + \mathrm{msd}(y) \geqslant n + v$ according to the definition of p_x and p_y, thus

$$\frac{|x_{p_x}| + |y_{p_y}|}{B^{p_x+p_y-n}} \leqslant \frac{4B}{B^v} = \frac{4}{B^{v-1}}$$

According to the definition of v, we have

$$\frac{4}{B^{v-1}} \leqslant \frac{1}{2}$$

and the case $p_y \geqslant \mathrm{msd}(y)$ is terminated.

Finally we consider the case if $p_y < \mathrm{msd}(y)$, then we have $|y_{p_y}| \leqslant 1$ and

$$\frac{|x_{p_x}| + |y_{p_y}|}{B^{p_x+p_y-n}} \leqslant \frac{2B\ B^{p_x-\mathrm{msd}(x)} + 1}{B^{p_x+p_y-n}} = \frac{2B\ B^n}{B^{p_y+\mathrm{msd}(x)}} + \frac{1}{B^{p_x+p_y-n}}$$

and we obtain the inequality

$$\frac{2B\ B^n}{B^{p_y+\mathrm{msd}(x)}} + \frac{1}{B^{p_x+p_y-n}} \leqslant \frac{2B}{B^v} + \frac{1}{B^{w-1}} = \frac{2}{B^{v-1}} + \frac{1}{B^{w-1}}$$

As before, according to the definition of v and w, we have

$$\frac{2}{B^{v-1}} + \frac{1}{B^{w-1}} \leqslant \frac{1}{2}$$

and the proof is complete. □

Algorithm 5 (*Inverse of a real number*). Let x be a real number respectively represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, we represent the inverse of this number $1/x$ by the sequence $(\overline{1/x}_n)_{n\in\mathbb{Z}}$ such that:

$$\text{If } n \leqslant -\mathrm{msd}(x) \text{ then } \overline{1/x}_n = 0 \text{ else } \overline{1/x}_n = \left\lceil \frac{B^{k+n}}{x_k + 1} \right\rceil + 1$$

$$\text{with } k = n + 2\mathrm{msd}(x) + w \text{ and } w = \begin{cases} 1 & \text{if } B \geqslant 3 \\ 2 & \text{if } B = 2. \end{cases}$$

Theorem 17. *If the computation of x terminates and x is not null, then the computation of $1/x$ terminates.*

4.3. Algorithms for algebraic or transcendental functions

4.3.1. General idea of these algorithms

For the computation of any algebraic or transcendental function f, we will use an intermediate function $\underline{f} : \mathbb{Q} \to \mathcal{R}$ such that for any rational number q for which $f(q)$ is

defined, $\underline{f}(q)_n$ satisfies the bounds property of $f(q)$ for any order n. We will then use these $\underline{f}$ functions to define the $\overline{f}$ functions but since the computation of $\underline{f}$ is more well known, even if this computation may lead to many tricks to be efficient, we will only quickly mention guidelines to compute them after this subsection to prove the completeness of our approach.

4.3.2. Algorithms

We present now such algorithms for algebraic and transcendental usual functions.

Algorithm 6 (*kth root*). Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$ and k be an integer greater than or equal to 2. We represent the kth root of this number $\sqrt[k]{x}$ by the sequence $(\overline{\sqrt[k]{x}}_n)_{n\in\mathbb{Z}}$ such that:

If $x_{kn} \geqslant 0$
then $\left\lfloor \underline{\sqrt[k]{x_{kn}}} \right\rfloor$
else fails.

Remark. We choose here to always give a value to $\sqrt[k]{x}$ when it make sense, even if it means that it does not fail for some slightly negatives values of x. We can of course choose to fail for all negative values by replacing the condition $x_{kn} \geqslant 0$ by $x_{kn} \geqslant 1$, but in this case for some slightly positive values it will fail also.

Algorithm 7 (*Exponential function*). Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, we represent $\exp(x)$ by the sequence $(\overline{\exp(x)}_n)_{n\in\mathbb{Z}}$ such that:

If $\underline{\exp}(x_m/B^m)_p \leqslant 0$, then $\overline{\exp(x)}_n = 0$ else
If $\overline{n > 0}$ and $\underline{\log_e}(1 - 1/B^n)_\ell + 2 < x_\ell < \underline{\log_e}(1 + 1/B^n)_\ell - 2$
or $n \leqslant 0$ and $\overline{x_0 \leqslant \lfloor \log_e(1 + 1/B^n) \rfloor - 1}$
then $\overline{\exp(x)}_n = B^n$
else $\overline{\exp(x)}_n = \left\lceil \left(\underline{\exp}(x_k/B^k)_p/B - 1 \right) (1 - 1/B^k) \right\rceil$

with $m = \max\left(0, \left\lceil \log_B(e/(B-1)) \right\rceil\right)$, $\ell = n + w$,

$$p = \max(0, \ell),\ c = \begin{cases} 2B & \text{if } x_{\mathrm{msd}(x)} \geqslant 1 \\ -1 & \text{otherwise} \end{cases},\ w = \begin{cases} 2 & \text{if } B = 2 \text{ or } B = 3 \\ 1 & \text{if } B \geqslant 4 \end{cases}$$

$$k = \begin{cases} \left(0, \mathrm{msd}(x),\ p + 1 + \left\lceil d + \log_B(e+1) \right\rceil\right) & \text{if } B = 2 \\ \left(0, \mathrm{msd}(x),\ p + 1 + \left\lceil d + \log_B((e+1)/(B-2)) \right\rceil\right) & \text{otherwise} \end{cases}$$

$$d = \max(-p, c\ \log_B(e)/B^{\mathrm{msd}(x)}),\ w = \begin{cases} 2 & \text{if } B = 2 \text{ or } B = 3 \\ 1 & \text{if } B \geqslant 4 \end{cases}$$

Remarks

(1) The first test aims to determine if x is a sufficiently negative number such that $\exp(x)$ may be approximated by 0 within B^{-n} rather than bother the multiplication of inequalities.

(2) The second test, corresponding to the double hypothesis, aims to determine if x is close enough to 0 such that $\exp(x)$ may be approximated by 1 rather than not to terminate in 0 because of the computation of msd(0).
(3) The natural logarithms $\log_e(z)$, where $z = 1 \pm 1/B^n$, are greater than $\underline{\log_e}(z)_0 - 1$ and less than $\pm 1/B^n$. We compute in the same way $\log_B(e/(B-1))$. Function $\underline{\log_e}$ refers to the logarithm function for rational arguments that we will suppose it exists in the next algorithm.
(4) Practically, we substitute the values of $\log_B e$ and $\log_B((e+1)/(B-2))$ by a simple upper or lower bound in the formula above (for example if $B = 4$, we use the fact that $0.72134 < \log_B(e) < 0.72135$ and $\log_B((e+1)/(B-2)) < 0.44732$).
(5) Practically, we can improve this algorithm by using the best known approximation of x after the determination of $\mathrm{msd}(x)$ and replace $d/B^{\mathrm{msd}(x)}$ by $(x_{\mathrm{mpa}(x)} + 1)/B^{\mathrm{mpa}(x)}$, so we obtain a finer upper bound and may appreciably reduce the value of k.

Algorithm 8 (*Logarithm to base B'*). Let B' be a real number greater or equal to 2 and x be a strictly positive real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, $\log_{B'}(x)$ is represented by the sequence $(\overline{\log_{B'} x}_n)_{n\in\mathbb{Z}}$ such that:

$$\overline{\log_{B'} x}_n = \left\lfloor (\underline{\log_{B'}}(x_k/B^k)_{n+w} + 1)/B^w + \log_{B'}(e)\, B^n/x_k \right\rfloor$$

$$\text{with } k = n + \mathrm{msd}(x) + w,\ w = c - \min(0, n) \text{ and } c = \begin{cases} 2 & \text{if } B \geqslant 3 \\ 3 & \text{if } B = 2. \end{cases}$$

Remarks
(1) The computation terminates if x is a finite strictly positive real number.
(2) Practically, we implement $\underline{\log_{B'}}$ only for the case $B' = e$ and if we want to compute the logarithm in another base B' of a real number, we deduce $\underline{\log_{B'}}$ from $\underline{\log_e}$ and then the corresponding function $\overline{\log_{B'}}$ or we deduce $\overline{\log_e}$ and then the function $\overline{\log_{B'}}$.

Algorithm 9 (*Arctangent*). Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, $\arctan(x)$ is represented by the sequence $(\overline{\arctan(x)}_n)_{n\in\mathbb{Z}}$ such that:

$$\text{If } x_k = \underline{0}$$
$$\text{then } \overline{\arctan(x)}_n = 0$$
$$\text{else } \overline{\arctan(x)}_n = \left\lfloor \frac{(\underline{\arctan}(x_k/B^k)_{n+w} + 1)}{B^w} + \frac{B^{n+k}}{B^{2n+2} + x_k^2 + x_k} \right\rfloor$$
$$\text{with } k = \max(0, n + w) \text{ and } w = \begin{cases} 1 & \text{if } B \geqslant 4 \\ 2 & \text{if } B = 2 \text{ or } B = 3. \end{cases}$$

Remarks. We can deduce π for example by the following formula due to Gauss:

$$\frac{\pi}{4} = 12\ \arctan\left(\frac{1}{18}\right) + 8\ \arctan\left(\frac{1}{57}\right) - 5\ \arctan\left(\frac{1}{239}\right).$$

4.3.3. Sine function

To complete our algorithmic we should be able to compute trigonometric functions. We have to choose between tan or sin (*vs*. cos) functions. On the one hand, after reduction of its

argument such that its absolute value would be lesser than $\pi/2$, tan is a monotonic function but the computation of $\underline{\tan}$ is uneasy and we will have to compute $\underline{\sin}$ and to deduce $\underline{\tan}$ from this one. On the other hand sin is not uniformly monotonic over $[-\pi/2, \pi/2]$ but is monotonic on two sub-intervals so it can be managed even if it is not so simple as for the previous functions, and $\underline{\sin}$ is easy. We have chosen to describe the sine function because this approach will be more efficient (with the first approach, if you compute $\overline{\sin(x)}$, you will have to deduce it from $\overline{\tan(x/2)}$ that you deduce from $\underline{\tan(x/2)}$ that is finally deduced from $\underline{\sin(x/2)}$) and this different example with an additional difficulty seemed to be more interesting.

Algorithm 10 (*Sine function*). Let x be a real number represented by the sequence $(x_n)_{n\in\mathbb{Z}}$. We note $p = \overline{\left(\frac{x}{\pi}\right)}_0 - 1$, $\theta = x - p\pi$ and $z = \frac{\pi}{2}$. We represent $\sin(x)$ by the sequence $(\overline{\sin(x)}_n)_{n\in\mathbb{Z}}$ such that:

$$
\begin{array}{l}
\text{If } 0 \leqslant \theta_k \leqslant 1, 4z_k - 4 \leqslant \theta_k \leqslant 4z_k + 4 \text{ or } 2z_k - 2 \leqslant \theta_k \leqslant 2z_k + 2,\\
\quad \text{then } \overline{\sin(x)}_n = 0\\
\quad \text{else if } 2 \leqslant \theta_k \leqslant z_k - 2 \text{ or } z_k + 2 \leqslant \theta_k \leqslant 2z_k - 3,\\
\qquad \text{then } \overline{\sin(x)}_n = (-1)^p \left\lfloor \dfrac{\underline{\sin}\left(\dfrac{\theta_k}{B^k}\right)_{n+w} + 1}{B^w} + B^{n-k} \right\rfloor\\
\qquad \text{else if } z_k - 1 \leqslant \theta_k \leqslant z_k + 1,\\
\qquad\quad \text{then } \overline{\sin(x)}_n = (-1)^p B^n\\
\qquad\quad \text{else if } 2z_k + 3 \leqslant \theta_k \leqslant 3z_k - 4 \text{ or } 3z_k + 4 \leqslant \theta_k \leqslant 4z_k - 5,\\
\qquad\qquad \text{then } \overline{\sin(x)}_n = (-1)^p \left\lceil \dfrac{\underline{\sin}\left(\dfrac{\theta_k}{B^k}\right)_{n+w} - 1}{B^w} - B^{n-k} \right\rceil\\
\qquad\qquad \text{else if } 3z_k - 3 \leqslant \theta_k \leqslant 3z_k + 3,\\
\qquad\qquad\quad \text{then } \overline{\sin(x)}_n = (-1)^{p+1} B^n
\end{array}
$$

$$
\text{with } k = \max(c, n + w) \text{ and } (c, w) = \begin{cases} (2, 2) & \text{if } B \geqslant 3 \\ (3, 4) & \text{if } B = 2. \end{cases}
$$

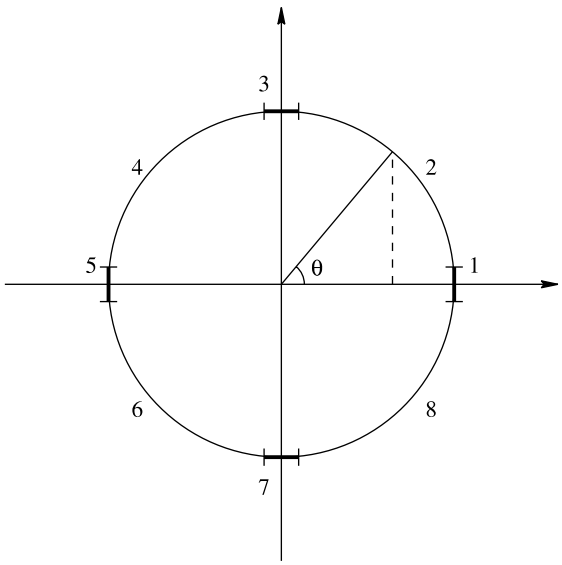
Remark. To manage the difficulty of the non uniform monotonicity of the sine function, we reduce the number to the equivalent between 0 and 2π, and then we carve the trigonometric circle in eight pieces numbered 1–8 in Fig. 1.

4.3.4. Other elementary functions

We deduce the other usual elementary functions from the preceding algorithms using the following formulas:

$$
\sinh(x) = \frac{\exp(x) - \exp(-x)}{2}, \quad \cosh(x) = \frac{\exp(x) + \exp(-x)}{2},
$$

$$
\tanh(x) = \frac{\sinh(x)}{\cosh(x)}, \quad x^y = \exp\left(\frac{y \log_B(x)}{\log_B(\exp(1))}\right), \quad \log_x y = \frac{\log_B y}{\log_B x},
$$



Legend for the zones:

zone 1 : $0 \le \theta_k \le 1$ *or* $4\,z_k - 4 \le \theta \le 4z_c + 4$, *sin*($\theta$) *is "closed" to* 0
zone 2 : $2 \le \theta_k \le z_k - 2$, *the sine function is increasing and positive "around"*θ
zone 3 : $z_k - 1 \le \theta_k \le z_k + 1$, sin($\theta$) *is "closed"to 1*
zone 4 : $z_k + 2 \le \theta_k \le 2z_k - 3$, *the sine function is decreasing and positive "around"*θ
zone 5 : $2z_k - 2 \le \theta_k \le 2z_k + 2$, sin($\theta$) *is "closed" to* 0
zone 6 : $2z_k + 3 \le \theta_k \le 3z_k - 4$, *the sine function is decreasing and negative "around"*θ
zone 7 : $3z_k - 3 \le \theta_k \le 3z_k + 3$, sin($\theta$) *is "closed" to* –1
zone 8 : $3z_k + 4 \le \theta_k \le 4z_k - 5$, *the sine function is increasing and negative "around"*θ.

Fig. 1. The eight intervals of the trigonometric circle for the computation of the sine function.

$$\operatorname{arcsinh}(x) = \log(x + \sqrt{x^2+1}), \quad \operatorname{arccosh}(x) = \log(x + \sqrt{x^2-1}),$$
$$\operatorname{arctanh}(x) = \frac{1}{2}\log\left(\frac{1+x}{1-x}\right),$$
$$\arcsin(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right), \quad \arccos(x) = \arctan\left(\frac{\sqrt{1-x^2}}{x}\right),$$
$$\cos(x) = \sin\left(\frac{\pi}{2} - x\right), \quad \tan(x) = \frac{\sin(x)}{\cos(x)}.$$

4.4. Comparison algorithms for real numbers

We use the expression *absolute comparison* for comparison that may loop for equal numbers but returns always exact results and *relative comparison* for comparison that never loops but returns only results within a given precision.

Algorithm 11 (*Absolute comparison between two real numbers*). Let x and y be two real numbers represented respectively by the sequences $(x_n)_{n\in\mathbb{Z}}$ and $(y_n)_{n\in\mathbb{Z}}$, then the result cmp(x, y) of the comparison between x and y is determined as follows:

$n = 0$

While $x_n + 1 > y_n - 1$ and $y_n + 1 > x_n - 1$ do $n \leftarrow n + 1$

If $x_n + 1 \leqslant y_n - 1$ then $\mathrm{cmp}(x, y) = -1$ else $\mathrm{cmp}(x, y) = 1$

Theorem 18

(1) *The algorithm terminates if and only if* $x \neq y$.
(2) $x < y$ *if and only if* $\mathrm{cmp}(x, y) = -1$.
(3) $x > y$ *if and only if* $\mathrm{cmp}(x, y) = 1$.

Algorithm 12 (*Relative comparison between two real numbers*). Let x and y be two real numbers represented respectively by the sequences $(x_n)_{n\in\mathbb{Z}}$ and $(y_n)_{n\in\mathbb{Z}}$, let k be an integer, then the result $\mathrm{cmp}_\varepsilon(x, y, k)$ of the comparison between x and y within a precision of B^{-k} is determined as follows:

$n = 0$

While $x_n + 1 > y_n - 1$ and $y_n + 1 > x_n - 1$ and $n \leqslant k + 2$ do $n \leftarrow n + 1$

If $x_n + 1 \leqslant y_n - 1$ then $\mathrm{cmp}_\varepsilon(x, y, k) = -1$

If $x_n - 1 \geqslant y_n + 1$ then $\mathrm{cmp}_\varepsilon(x, y, k) = 1$

else $\mathrm{cmp}_\varepsilon(x, y, k) = 0$

Theorem 19

(1) *The algorithm always terminates.*
(2) *We have* $\mathrm{cmp}_\varepsilon(x, y, k) = -1$ *only if* $x < y$.
(3) *We have* $\mathrm{cmp}_\varepsilon(x, y, k) = 1$ *only if* $x > y$.
(4) *We have* $\mathrm{cmp}_\varepsilon(x, y, k) = 0$ *if and only if* $|x - y| < \frac{1}{B^n}$.

Remark. These theorems suppose that the computation of x_n and y_n terminates.

4.5. Existence of the $\underline{f}$ functions for all the f functions mentioned above

We assume at the beginning of the previous section that for any "basic" transcendental function f there exists a function $\underline{f} : \mathbb{Q} \to \mathcal{R}$, that maps any rational number r to a sequence $\underline{f}(r, n)/B^n$ ($\underline{f}(r, n) \in \mathbb{Z}$) such that

$$\frac{\underline{f}(r, n) - 1}{B^n} < f(r) < \frac{\underline{f}(r, n) + 1}{B^n}.$$

k-root is a special case since the function $x \mapsto \underline{\sqrt[k]{x}}$ is defined directly on $\mathbb{N}$ and maps any integer x to $\lfloor \sqrt[k]{x} \rfloor$. Such a function is computed by Newton's method applied to the function $z \mapsto z^n - x$ (see [21] for more details).

4.5.1. Basic case for transcendental functions

The main argument of our proof for the existence of such a function $\underline{f}$ is that this function is defined by an alternating series converging on a non empty interval

$$f(r) = s = \sum_{i\in\mathbb{N}} (-1)^i a_i$$

where $(a_i)_{i\in\mathbb{N}}$ (if the general term of the Taylor expansion of f is b_n, $a_n = b_n r^n$) is a sequence of rational numbers with same common sign.

This approach has two advantages: first of all, we have a very simple stop condition for the series converging according to the alternating series criterion since it is sufficient that $|a_{n+1}|$ is lesser than the required error ε to ensure that the sum of this series up to rank n

$$s_k = \sum_{i=0}^{i=k} (-1)^i a_i$$

is an approximation of the limit s within ε. Furthermore, two consecutive terms of such a series supply an interval with rational bounds containing the limit: if all terms of the sequence $(a_i)_{i\in\mathbb{N}}$ are positive, then $s_{2i+1} < s < s_{2i}$ for any $i \in \mathbb{N}$ and $s_{2i} < s < s_{2i-1}$ for any $i \in \mathbb{N}^*$, if all terms of the sequence $(a_i)_{i\in\mathbb{N}}$ are negative.

So we have a recursively enumerable sequence of nested intervals with rational bounds including the real number to compute and with length vanishing to 0, that is to say that this real number is computable according to some definition and the equivalence of this notion with the notion of B-approximable real number we consider here supply us an algorithm to compute an integer λ (and a value for k to compute λ) such that λ satisfies the bound property of s for order n. Essentially we use the fact that $s_k = p_k/q_k$ and $q_k \geqslant B^n$ so $\lambda = \lfloor s_k B^n \rfloor$. For more details see [22].

Consequently, we will come to this simple case for each basic function and prove that this transformation preserves the bounds property.

4.5.2. Exponential function

Let r be a rational number, we represent $\exp(r)$ by the product of $e^{\lfloor r \rfloor + 1}$ by $\exp(r - (\lfloor r \rfloor + 1))$ if r is not null and 1 otherwise. The first term of this product is computed as a power of e, that is to say by successive multiplications (and inversion if r is negative). The computation of the second term uses the Taylor expansion of $\exp(x)$ for $-1 \leqslant x < 0$. The basic case can be directly applied to this term. The computation of e may be performed either by using directly the series of general term $1/n!$ and the inequality

$$\sum_{k\geqslant n+1} 1/k! < 1/(n\, n!)$$

for the stop condition and an interval with rational bounds including e: $s_n < e < s_n + 1/(n\, n!)$ with the same usage as above, or by inversion of $\exp(-1)$, computed according to the second case. Whatever the choice, we use the multiplication of two real numbers and possibly the inversion of a real number, and we obtain function exp.

4.5.3. Logarithm function

Let r be a rational number, we compute $\ln(r)$ as follows: if $r \leqslant 0$, then the computation fails; if $r < 1$, then we take $\ln(r) = -\ln(1/r)$; if $r = 1$, then the result is the null sequence; if $r > 1$, then we use the formula

$$\ln(r) = 2 \operatorname{arctanh}\left(\frac{r-1}{r+1}\right)$$

with $y = (r-1)/(r+1)$, we have

$$\operatorname{arctanh}(y) = \sum_{k\geqslant 0} \frac{y^{2k+1}}{2k+1}$$

and

$$\sum_{k \geqslant n+1} \frac{y^{2k+1}}{2k+1} \leqslant \frac{y^{2n+3}}{(2n+3)\,(1-y^2)}.$$

Thus we have an interval with rational bounds $s_n < \ln(r) < s_n + y^{2n+3}/((2n+3)(1-y^2))$ and we use it as for above.

4.5.4. Trigonometric functions

The Taylor expansion of $\arctan(r)$ and $\cos(x)$ are alternating series for any rational x. The Taylor expansion of $\sin(x)$ is an alternating series for any positive rational x and the sine function is odd so we can always boil down to the alternating series.

4.5.5. Remark

This is a proof of the existence of at least one set of functions $\underline{f}$ but it is only one possible way to compute one such set. For example, we can also use ideas similar to those described by Brent in [9] to compute these functions.

5. Implementation

5.1. The choice of the Caml language

The use of this language was of course a natural choice insofar as we began the study on the subject of arithmetic for a modern and reliable programming language about this language. Furthermore the first step of this study leads us to implement a very efficient exact rational arithmetic for this language, that relies on the Bignum package [23,24].

It is obvious that (almost) infinite integers are absolutely necessary, but an exact rational arithmetic (see [25]) is also necessary to compute the transcendental functions on rational parameters underlying the transcendental functions on real arguments.

Moreover functions in this language are easy to use as arguments or results of functions and since real numbers (and more generally infinite objects) are naturally represented by functions, it is easier to deal with real numbers in this language.

5.2. Choices of implementation

We choose as Boehm to represent real numbers as finite B-adic numbers and furthermore these particular finite B-adic numbers, instead as general rational numbers. This choice leads us to a rougher granularity and a slightly lesser flexibility for our representation. For instance, if an accuracy under $1/B^n$ is required, this choice of implementation leads to a computation with an accuracy of $1/B^{n+1}$ and induce a greater running time than a computation to the real precision of $1/B^n - \varepsilon$ where ε is a rational number as small as possible.

Boehm's implementation used rational numbers at the beginning and it turned out that with the library of rational arithmetic used by Boehm, the computations with rational numbers were much slower than those performed with finite B-adic numbers, so he finally

choose B-adic numbers to represent real numbers. But this choice deprive us of the natural incrementality of the representation and of a slightly simpler expression of our algorithms. Indeed this representation, if we do not use the mpa functionality, is not incremental, that is to say that if we have computed a result to n digits, if we want to compute its value to $n+1$ digits we need to compute it again from scratch. But we adapt the representation to lessen this drawback by the following choices.

The implementation includes the storage of the most precise approximation already computed for each real number and we choose to work with the base $B = 4$. We will now justify our choice.

For efficiency, the representation includes for each real number x represented by the sequence $(x_n)_{n\in\mathbb{Z}}$, not only the functional closure but also the most precise approximation already computed $x_{\mathrm{mpa}(x)}$ to the order $\mathrm{mpa}(x)$ as mentioned above in page 20. This choice leads us to redo only partly the computation: for example with a function that consider the size of its argument(s) with the msd function, these arguments have been computed to a precision sufficient to compute their msd and the computation of msd is reduced after this first computation to some shifts operations and maybe the already computed approximations for part of the arguments will be sufficient.

Practically a good sharing of the expressions will increase the effect of this information storage and reduce the time of computation.

Concerning the choice of the base, it is preferable to choose 2 to some power, since

$$\left\lfloor \frac{x_{\mathrm{mpa}(x)}}{B^{\mathrm{mpa}(x)-n}} \right\rfloor$$

can be computed by a simple computer shift of $x_{\mathrm{mpa}(x)}$ of $\mathrm{mpa}(x) - n$ digits to the right in this case, that is to say a basic operation of rational arithmetic and then for the underlying hardware arithmetic. It may seem worthwhile that a digit for the base B corresponds exactly to a computer word. However we have also to consider that the smaller the base is and the less we pay to compute an additional digit. Hans Boehm choose to work with $B = 4$ and so do I. This is a good compromise between on the one hand the trend to perform computations on computer words and on the other hand the fact that if we need to compute one digit more for a number the cost should not be very different.

5.3. Realizations

Boehm implemented a similar arithmetic, so one can read his commentaries in [13,14]. Moreover we have currently a complete prototype for this representation. Tests are in progress. The chosen representation has the advantage of using algorithms on integers that are well understood and very efficient.

6. Conclusion

We have a description of a representation of $\mathcal{R}$ and proved algorithms for this representation for all elementary functions.

We have implemented a complete prototype of this description so we have a complete chain for reliable arithmetic in the Caml language.

In the future, it may be also interesting to study the influence of the radix B on the efficiency of real computations. Furthermore we hope to improve the efficiency of this prototype by an optimized computation of the $\underline{f}$ functions with optimizations like those

developed in [26] and by a balancement of the abstract syntax tree during the compilation of expressions such as $x_1 + \cdots + x_n$ to compute each x_i with a well-balanced precision.

It seems to be interesting to combine our arithmetic with floating point analysis method such as interval analysis [5,6,7,8] or the CESTAC method [27,28,29] in the spirit of the lazy rational arithmetic by Michelucci [30]: it consists in computing the functional closure of the result and at the same time to compute the interval result according to interval analysis. If a result (maybe an intermediary result) is not precise enough we compute it with exact real arithmetic at the needed precision. This method has the advantage that it is efficient and precise: generally speaking, big floating point applications accept to pay in time only when necessary so this solution is well adapted to this need. However this approach is limited by the fact that IEEE floating point standard arithmetic concerns only rational operations currently.

Our goal is not to substitute our arithmetic to floating-point arithmetic, that is often sufficient and very efficient, but to make available an alternative arithmetic for specific needs. We want to be able to compute a reliable result even if it takes a while rather than to obtain a wrong result immediately.

An idea consists to consider this arithmetic as a static analysis of the needed precision for a floating point computation according to the required precision on the result.

Finally, it would be interesting to build real analysis on top of this real arithmetic.

7. Current state of the art

We summarise some relevant recent work here, with correctness results or not.

7.1. Continued fractions, Möbius transformations, LFT

David Lester in [31] describes a type of continued fractions for which Gosper'algorithms are correct.

Peter Potts and Abbas Edalat in [32–34] represent real numbers as Linear Fractional Transformations (LFT) and show how to encode continued fractions using LFT and deduce algorithms to compute with LFT. Reinhold Heckmann in [35–37] shows how to manage computations with LFT according to the expected precision.

7.2. Computable Cauchy sequences

David Lester and Paul Gowland in [38] presents an arithmetic using effective Cauchy sequences (sequences of finite 2-adic numbers) with algorithms similar to ours for rational operations (including iterators), square root and simplistic transcendental functions using power series.

7.3. Adaptive computations

This approach consists in an iterative bottom-up analysis. The computation starts with a predefined precision on all inputs and at each step of the computation, if the required precision is not obtained, the computation is performed with increased precision.

MPFR (Polka team at INRIA Loria, directed by Paul Zimmermann [39]) computes with floating-point representations.

The iRRAM work of Norbert Müller [40] relies on the REAL RAM by Vasco Brattka and Peter Bretling [41].

7.4. Implementations

Jean Vuillemin have carried out a small implementation of continued fractions in Lisp, but it never was available in no way at all.

Hans Boehm have implemented a pocket calculator and a version in Java is currently available at `http://www.hpl.hp.com/personal/Hans_Boehm/crcalc/CRCalc.html`. We mentioned in Section 5.3 three implementations of our work.

Peter Potts have made a small prototype for LFT in Caml named Calathea, available at `http://www.purplefinder.com/~potts/calathea.zip`. A complete prototype named IC-reals in C is available at `http://www.doc.ic.ac.uk/~ae/ic-reals-6.2-beta.tar.gz`.

David Lester in [31] mentions a Haskell very slow implementation using continued fractions and a more classical computable Cauchy sequences representation used in the implementation MAP presented in [38,42], available at `http://www.cs.man.ac.uk/arch/dlester/exact.html` (Haskell version, C version announced).

Norbert Müller has written the C++ very efficient package iRRAM available at `http://www.informatik.uni-trier.de/iRRAM/`.

Paul Zimmermann et al. make MPFR [43] available at `http://www.loria.fr/projets/mpfr/`.

Paul Gowland and David Lester have surveyed exact real arithmetic implementations in [44] and Jens Blanck have compared them in [42].

There are two implementations of this work: CREAL written in OCaml by Jean-Christophe Filliâtre [45] and XR written in python, C++ and C by Keith Briggs [46–48].

7.5. Mechanically checked proofs

In [49], David Lester and Paul Gowland have proved in PVS the correctness of their algorithms on computable Cauchy sequences described in [38], relying on the NASA Langley PVS real library for axiomatic definitions of the transcendental functions. The complete proof is available at `http://www.cs.man.ac.uk/arch/dlester/exact.html`.

David Lester in [31] mentions machine-assisted proofs for the central algorithms for rational operations on continued fractions.

Jérôme Créci in [50] has defined our representation and proved our addition, subtraction and multiplication algorithms in Coq, relying on the Reals library axiomatized in the Coq system. When all our algorithms will be proved in Coq, we will be able to combine this work with the real analysis available in the axiomatization of real numbers.

Paul Zimmermann proved some algorithms of MPFR in Coq with the help of Lemme team of Inria Sophia-Antipolis.

Acknowledgements

This work took place essentially at INRIA Rocquencourt during the Ph.D. of the author and minorly at Université d'Évry-Val d'Essonne.

The author thanks the Theory and Formal Methods at Imperial College for his warm hospitality during the major step of the writing of this article, supported at Imperial College by EPSRC project "Techniques for real number computation".

Jean-Michel Muller advises me all along this publication work. Moreover his relevant questions at the time of my PhD defense made me aware of the additional work to do for transcendental functions on rational numbers.

The formal proof in Coq of Jérôme Créci reveals a small error in multiplication and now this algorithm is correct and formally proved. I hope that this work of formalization and automated proof will be achieved.

Jean-Christophe Filliâtre suggests an improvement for the computation of the sin function, make its CREAL implementation of our work available and trained Jérôme Créci during his master thesis's work.

References

[1] D. Goldberg, What every computer scientist should know about floating point arithmetic, ACM Computing Surveys, 23 (1) (1991), 5–47, Available from: <http://docs.sun.com/htmlcoll/coll.648.2/iso-8859-1/NUMCOMPGD/ncg_goldberg.html>.
[2] J.-M. Muller, Arithmétique des ordinateurs, Etudes et recherches en informatique, Masson, 1989.
[3] J.-M. Muller, Ordinateurs en quête d'arithmétique, La Recherche 26 (278) (1995) 772–777.
[4] J.-M. Chesneaux, L'approche probabiliste des erreurs d'arrondi, Talk to a workshop on the quality of computers results organized by Paris 6 University, April 1997.
[5] K. Nickel (Ed.), Interval Mathematics, lNCS, vol. 29, 1975.
[6] K. Nickel (Ed.), Proceedings of the International Symposium on Interval Mathematics, lNCS, vol. 212, Springer-Verlag, 1985.
[7] O. Aberth, Precise Numerical Analysis, Wm.C. Brown Publishers, 1988.
[8] M. Daumas, C. Mazenc, J.-M. Muller, User transparent interval arithmetic, in: Proceedings of IMACS/GAMM International Symposium SCAN-93, 1993. Also available as research report number 94-02 of École Normale Supérieure de Lyon, January 1994. Available from: <ftp://ftp.lip.ens-Lyon.fr/pub/Rapports/RR/RR94/RR94-02.ps.Z>.
[9] R.P. Brent, Fast multiple-precision evaluation of elementary functions, Journal of the ACM 23 (2) (1976) 243–251.
[10] E. Wiedmer, Exaktes R echnen mit reellen Zahlen und anderen unendlichen Objekten, Ph.D. thesis, ETH, Zurich, diss. ETH 5975, 1977.
[11] E. Wiedmer, Calculs avec des fractions décimales et d'autres objets infinis, Compte-rendu d'un exposé à l'Institut de Programmation de l'Université Paris VII, January 1979.
[12] E. Wiedmer, Computing with infinite objects, Theoretical Computer Science 10.
[13] H.J. Boehm, R. Cartwright, M.J. O'Donnel, M. Riggle, Exact real arithmetic: a case study in higher order programming, in: Proceedings of the 1986 ACM Conference on Lisp and Functional Programming, ACM, 1986.
[14] H.J. Boehm, Constructive real interpretation of numerical programs, in: Proceedings of the 1987 ACM Conference on Interpreters and Interpretives Techniques, ACM, 1987.
[15] A. Avizienis, Signed-digit number representations for fast parallel arithmetic, IRE Transactions on Electronic Computers (10) (1961) 389–400.
[16] H.X. Lin, H.J. Sips, On-Line Cordic Algorithms, IEEE Transactions on Computers 39 (8) (1990) 1038–1052.
[17] J. Vuillemin, Exact real computer arithmetic with continued fractions, Research report Rapport de recherche 760, INRIA, 1987.
[18] J. Vuillemin, Exact real computer arithmetic with continued fractions, in: Proceedings ACM Conference on Lisp and Functional Programming, ACM, 1988, Extended version as INRIA research report 760, 1987.
[19] J. Vuillemin, Exact real computer arithmetic with continued fractions, IEEE Transactions on Computers 39 (8) (1990) 1087–1105.

[20] B. Gosper, Continued Fraction Arithmetic, HAKMEM Item 101B, MIT AI MEMO 239, February 1972. Available from <ftp://ftp.netcom.com/pub/hb/hbaker/hakmem/cf.html#item101b>.
[21] V. Ménissier-Morain, Arithmétique exacte, conception, algorithmique et performances d'une implémentation informatique en précision arbitraire, Thèse, Université Paris 7, December 1994. Available from <http://calfor.lip6.fr/vmm/documents/these94.ps.gz>.
[22] V. Ménissier-Morain, Arbitrary precision real arithmetic: design and proved algorithms, Research report Rapport de recherche 2003.003, Université Paris 6, January 2003. Available from <http://calfor.lip6.fr/vmm/documents/rapport2003.ps.gz>.
[23] J.-C. Hervé, F. Morain, D. Salesin, B. Serpette, J. Vuillemin, P. Zimmermann, BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic, Tech. Rep. 1016, INRIA, Domaine de Voluceau, 78153 Rocquencourt, FRANCE, April 1989.
[24] B. Serpette, J. Vuillemin, J. Hervé, BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic, Tech. Rep. 2, Digital PRL, May 1989.
[25] V. Ménissier-Morain, The CAML Numbers Reference Manual, Tech. Rep. 141, INRIA, July 1992. Available from <http://calfor.lip6.fr/vmm/documents/doc_arith_without_camels.ps.gz>.
[26] C. Batut, Aspects algorithmiques du système de calcul arithmétique en multiprécision PARI, Thèse de doctorat, Université de Bordeaux I, February 1989.
[27] J. Vignes, M. La Porte, Error analysis in computing, Information Processing, North-Holland, 1974
[28] J.-M. Chesneaux, Study of the computing accuracy by using probabilistic approach, in: C. Ullrich (Ed.), Contribution to Computer arithmetic and Self-Validating Numerical Methods, Academic Press, 1990, pp. 19–30.
[29] J. Vignes, A stochastic arithmetic for reliable scientific computation, Mathematics and Computers in Simulation 35 (1993) 233–261.
[30] M. Benouamer, P. Jaillon, D. Michelucci, J.-M. Moreau, A lazy solution to imprecision in computational geometry, in: Proceedings of the 5th Canadian Conference on Computational Geometry, 1993, pp. 73–78. Available from <ftp://ftp.emse.fr/pub/papers/LAZY/LazyCG.ps.gz>.
[31] D. Lester, Effective continued fractions, in: Proceedings of the 15th IEEE Symposium on Computer Arithmetic, 2001, pp. 163–170.
[32] A. Edalat, P.J. Potts, A new representation for exact real numbers, Electronic Notes in Theoretical Computer Science 6 (14).
[33] P.J. Potts, Exact real arithmetic using Möbius transformations, Ph.D. thesis, Imperial College, 1999.
[34] P.J. Potts, A. Edalat, P. Sünderhauf, Lazy computation with exact real numbers, in: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming, 1998, pp. 185–194.
[35] R. Heckmann, Contractivity of linear fractional transformations, Theoretical Computer Science 279 (1) (2002) 65–82., third Real Numbers and Computers Conference (1998).
[36] R. Heckmann, How many argument digits are needed to produce *n* result digits, Electronic Notes in Theoretical Computer Science 24, realComp '98 Real Number Computation, Indianapolis, June 1998.
[37] R. Heckmann, Translation of Taylor series into LFT expansions, in: Proceedings of Dagstuhl Seminar Symbolic Algebraic Methods and Verification Methods, November 1999, submitted for publication.
[38] P. Gowland, D. Lester, The correctness of an implementation of exact arithmetic, in: Proceedings of the Fourth Conference on Real Numbers and Computers, 2000.
[39] P. Zimmermann, MPFR: a library for multiprecision floating-point arithmetic with exact rounding, in: Proceedings of the Fourth Conference on Real Numbers and Computers, 2000, pp. 8–90.
[40] N.T. Müller, The iRRAM: exact arithmetic in C++, in: Computability and Complexity in Analysis, Vol. 2064 of Lecture Notes in Computer Science, 2001, pp. 222–252, Proceedings of the 4th International Workshop, CCA 2000, September 2000. Available from <http://www.informatik.uni-trier.de/mueller/>.
[41] V. Brattka, P. Hertling, Feasible real random access machines, Journal of Complexity 14 (4) (1998) 490–526.
[42] J. Blanck, Exact real arithmetic systems: results of competition, Computability and Complexity in Analysis, Lecture Notes in Computer Science, vol. 2064, Springer, 2001, pp. 389–393, 4th International Workshop, CCA 2000, September 2000.
[43] MPFR Team, LORIA, INRIA Lorraine, MPFR: Multiple Precision Floating-Point Reliable Library, second ed., April 2002. Available from: <http://www.loria.fr/projets/mpfr/mpfr-current/documentation.html>.
[44] P. Gowland, D. Lester, A survey of exact arithmetic implementations, in: Computability and Complexity in Analysis, Lecture Notes in Computer Science, vol. 2064, 2001, pp. 30–47, 4th International Workshop, CCA 2000, September 2000.

[45] J.-C. Filliâtre, Description of the Creal module for Objective Caml , World-Wide Web document, November 2001. Available from: <http://www.lri.fr/filliatr/ftp/ocaml/ds/creal.ps.gz>.
[46] K. Briggs, Y. Smaragdakis, XR—exact real arithmetic, World-Wide Web document and software package, March 2001. Available from: <http://www.btexact.com/people/briggsk2/XR.html>.
[47] K. Briggs, Exact real computation, Talk at University of Warwick, May 2001. Available from: <http://www.btexact.com/people/briggsk2/xr-talk.ps>.
[48] K.M. Briggs, Implementing exact arithmetic in python, C++ and C, submitted to the special issue of TCS dedicated to RNC5, accepted for publication.
[49] D. Lester, P. Gowland, Using PVS to validate the algorithms of an exact arithmetic, Theoretical Computer Science 291 (2) (2002) 203–218, Available from: <http://www.cs.man.ac.uk/arch/dlester/exact.html>.
[50] J. Créci, Certification d'algorithmes d'arithmétique rélle exacte dans le système Coq, DEA Logique et Fondements de l'Informatique , Université Paris 11, 2002.