Efficient Exact Arithmetic over Constructive Reals

Yong Li, Jun-Hai Yong

Efficient Exact Arithmetic over Constructive Reals

Yong Li and Jun-Hai Yong

School of Software, Tsinghua University, Beijing 100084, P.R. China
exactreal@yahoo.com.cn

Abstract. We describe a computing method of the computable (or constructive) real numbers based on analysis of expressions. This method take precision estimate into account in order to get a better algorithm than Ménissier-Morain's method, which is also based on the representation of constructive reals. We solve two problems which appear in exact real arithmetic based on the representation of constructive reals. First, by balancing every item's precision in the expression, we can avoid unnecessary precision growth. Second, by distributing different weights to different operations, we can make sure that complex operations do not waste much time when to compute the whole expression. In these ways, we finally get a more efficient and proper method than prior implementations.

1 Introduction

The goal of exact arithmetic is to generate correct answers to numeric problems, within some user-specified error. Generally speaking, floating-point numbers are widely used to represent real numbers in computer systems, although the limitations are quit obvious. For instance, the real numbers cannot be arbitrarily large or small, and the rounding errors can lead to inaccurate or completely wrong results. Some examples can be found in [1,2]. In order to solve these limitations, many methods have been proposed, one of which is exact arithmetic that has attracted more and more attentions in recent years.

Several methods have been developed for exact arithmetic which can supply an approximation of the exact result that meets the user's any requirement of precise. V. Ménissier-Morain in [3] gives a brief introduction of these methods. However, only two of them are practical and proved completely correct. One is linear fractional transformation (LFT) which is mostly based on continued fractions. In 1976, Gosper first introduced this mthod to calculate the operations of real numbers [4]. In 1989, Vuillemin implemented the arithmetic using Lisp and improved this arithmetic in many ways [5]. In 2002, Edalat and Heckmann developed this arithmetic to LFT Framework [1] and they also supply an implementation in Haskell and C, which is named IC-Reals. The other one , constructive reals, is based on Cauchy Sequences, a theory that is well established by Bishop[6]. It can also be considered as a foundation of computable analysis. Boehm and Lee first made use of this theory for the purpose of exact arithmetic [7,8] and Boehm finished this algorithm using java [9]. Gowland and Lester in

[10,11] showed us the correctness of an implementation based on fast binary Cauchy sequence, which makes the base equal to 2. And they have described a Haskell implementation in [10]. In addition, V. Ménissier-Morain in [3] proved a similar algorithm whose base can be 2 or other integers which is bigger than 2. Briggs implemented this paper's algorithm using Python, C++, C respectively [12]. Of course, there are also many other implementations and you can get more information about exact real arithmetic in this survey [13]. But most of them are deficient in the proof or integrality of algorithms.

Actually, all these methods can be converted into interval representations. But different methods have different algorithms to calculate the operation concerning real numbers. In that case their performances are also different. In terms of basic operations, like addition, minus, multiplication and division, the second method may be more efficient, which can be found in this competence result in 2000 [14]. However, LFT approach also has some merits that it can easily handle irrational numbers [15,16], that Edalat argues its transcendental functions may more efficient, and that under special circumstances continued fractions can solve many problems which common representations is hard to handle. In this paper, our interest focus on the amelioration of the second methods, that is constructive reals. And we give new algorithms that solve two problems which can affect the implementation's performance. First, by balancing every item's precision, we can avoid unnecessary precision growth which is the first problem that has attracted many attentions in [17,18]. Second, by distributing different weights to different operations, we can make sure that complex operations do not waste most time when to compute the whole expression which is the second problem that we proposed.

The remaining part of the paper is arranged as follows. Related definitions and algorithms about Cauchy sequences are briefly introduced in section 2. Problems of present implementations based on Cauchy sequences are shown in section 3. A new simple algorithm about addition, which considers balancing every item's precision, is proved in section 4. And a new algorithm about calculating the expression that takes every operation's complexity into consideration will be given in section5. The experimental result and discussion are set in section 6.

2 Basic Definitions and Algorithms

We present some basic definitions and algorithms of computable real numbers, which derived from Boehm, V. Ménissier-Morain, and Gowland's work in [3,9,10]. First, we will show two definitions about the representation of real numbers. Second, we will give two simple algorithms about addition and multiplication that will help us understand the problem.

Definition 1. *(Effective Cauchy sequences). A computable real number x is represented as an Effective Cauchy sequence, if there is an infinite computable sequence of rational numbers* $\left\{\frac{n_0}{d_0}, \frac{n_1}{d_1}, \ldots, \frac{n_p}{d_p}, \ldots,\right\}$*, with* $d_i > 0$*, and a modulus*

of convergence function $e : N \to N$ *which is recursive , such that* $\forall p \in \mathbf{N} : k \geq e(p)$*implies* $\left|x - \frac{n_k}{d_k}\right| < 2^{-p}$.

In this definition, we use a sequence of rational numbers to represent the real number. When we need to get a number which meets the precision requirement, we can take a proper rational number from the sequence. In fact, we can get an interval which is small enough to meet the error range and includes the true value of the real number that is $x \in \left(\frac{n_k}{d_k} - 2^{-p}, \frac{n_k}{d_k} + 2^{-p}\right)$. But if we only want to use integer arithmetic to finish this kind of exact real arithmetic, we should do some changes.

Definition 2. *(Fast Binary Cauchy Sequence) A computable real number* x *is represented as a Fast Binary Cauchy Sequence if there is an infinite computable sequence of integers*$\{n_0, n_1, \ldots, n_p, \ldots\}$*, such that* $|x - 2^{-p}n_p| < 2^{-p}$.

This definition makes a little change with respect to Definition 1. We can use a sequence of integers to represent real number, which makes implementation only use integer arithmetic and is not confined by floating-point numbers' defects. Some implementations using floating point, such as IRRAM [19], which is based on real-RAMs, have faster speed. Although some real-RAMs can be realized approximately by floating point computations, real-RAMs cannot be realized by physical machines, they are unrealistic [20]. But the way of construct reals is completely accepted in that it only uses integer arithmetic. So in section 5 we do not compare time with IRRAM which use floating-point arithmetic in implementation. In this definition, if the error range is 2^{-p}, we can use n_p to represent the real number x, which meets the requirement of $x \in (2^{-p}n_p - 2^{-p}, 2^{-p}n_p + 2^{-p})$. In the rest of the article, we use $x[p]$ to represent n_p.

Also, there are other similar definitions, in[3], they use the qualification of $\left|x - \frac{n_k}{d_k}\right| < B^{-p}$ where $B \geq 2$ is some fixed integer, instead of $\left|x - \frac{n_k}{d_k}\right| < 2^{-p}$ in Definition 1. Obviously, modulating B can control the interval's and the error range. For example, if B is 10, we can get a sequence of error range:10^{-1}, 10^{-2}, $\ldots$, 10^{-p}, $\ldots$.If B is 2, we can get a sequence of error range:2^{-1}, 2^{-2}, $\ldots$, 2^{-p}, $\ldots$which is more flexible than B is 10 and can easily meet more error requirements. In this paper, we only discuss our implementation based on $B = 2$ and these circumstances that $B > 2$ can be handled in the same thought.

Now we simply give some algorithms concerning addition and multiplication which is widely used in [3,9,10] and is showed in [10]. These algorithms can help us to illustrate our algorithms. Their proofs can be found in [10]. We suppose that x,x_1,x_2 are real numbers and the required error range of x is 2^{-p} which means we need to get $x[p]$.

Algorithm 1. *(Addition of real numbers) Computing* $x = x_1 + x_2$*, we have* $x[p]$=*round*$((x_1[p+2] + x_2[p+2])/4)$.

Algorithm 2. *(Multiplication of real numbers) Computing* $x = x_1 * x_2$*, we have* $x[p]$=*round*$(2^{-(p+s_1+s_2)}(n_1n_2))$ *where* $s_1 = \lfloor log_2(|x_1[0]| + 2)\rfloor + 3$ *and* $s_2 = \lfloor log_2(|x_2[0]| + 2)\rfloor + 3$.

3 Problems

Several authors have implemented the algorithms that are based on constructive reals. We can divide them into two kinds. One is to use Functional language which is easy and natural to implement exact real arithmetic in that we can regard a real number as a function and the expression can be evaluated automatically. However, this way is very slow and impractical in many applications. The other is to use DAG (directed acyclic graph) to represent the expression about operations between exact real numbers. For example, Figure 1 shows an expression based on DAG. This way can be finished by using C language, which can get much better performance and also is easier to use in other software. But both of them suffer from several flaws.

In the first problem, with the increase of operations' number, the original algorithms fail to use proper precise of every real number. For example, we want to compute $x = x_1 + x_2 + x_3 + x_4$ according to Algorithm 1. Its DAG is figure 1, and when we want to get $x[p]$, we need get $x_4[p + 2]$, $x_3[p + 4]$, $x_2[p + 6]$, $x_1[p + 8]$ respectively. Obviously, when the items' number is very large we may need some items that have extremely small precision. However, by scrutinizing this case carefully, we find it is not reasonable. If we consider it from a prospect of expression we find if we want to get an approximation of x up to the precision ϵ, we only need every item' approximation up to $\epsilon/4$. In this case, we should balance every item's precision to get a more proper means to compute the expression. This problem has been proposed in [17,18], which occurs in all exact real arithmetic based on Type2 theory [20]. In [18], the author proposed that each item's precision should be balanced to solve this problem. But in special representation how to solve it is also a question. In this paper we give the methods about constructive reals and give a proof in the follow section.

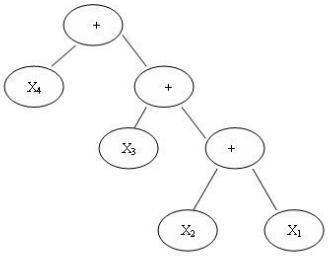

Fig. 1. A DAG for the expression $x_1 + x_2 + x_3 + x_4$

In the second problem, the original algorithm fails to take the complexity of each operation into consideration. Obviously, computing one multiplication takes more time than computing an addition when we need them to get the same precision which can be found in Algorithm 1 and Algorithm 2. The original algorithms

are not concerned about making the precision of operation of multiplication as large as possible. For instance, when we compute $x = x_1 + x_2 * x_3$, in order to get $x[p]$, a choice is to get $x_1[p+2]$ and $(x_2 * x_3)[p+2]$ which is an original algorithm's way. However, in terms of expression, we have many other ways. For example, when we want to get an approximation of x up to the precision ϵ, we also need x_1's approximation up to $\epsilon/4$ and $(x_2 * x_3)$'s approximation up to $3\epsilon/4$. In this case, the operation of addition will cost more time than the former and the operation of multiplication will cost little time than the former. But we think it is totally worthy because the multiplication is more complex.

In conclusion, original implementations do not consider the difference between exact real arithmetic and traditional computation. Traditional computation can use a common way to compute because they are not affected by error requirement. However, in exact arithmetic, distributing a proper precision to every item is very important which can make computation more effective.

4 Addition Algorithm with Balanced Precision

Addition and subtraction are very important in exact arithmetic in that summation plays an important role in science computing. And addition and subtraction's error analysis are also very easy to process. In this part we give an algorithm to compute an expression only including addition. Subtraction can be processed in a similar way in that it can be taken as a special addition.

Algorithm 3. *Computing $x = x_1 + x_2 + \ldots + x_n$, where x, x_1, x_2,..., x_n are real numbers and the required of error range of x is 2^{-p} which means we need to get $x[p]$, we have $x[p]$=round($(x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m}$). where $m = \lceil log_2 n + p + 1 \rceil$.*

Proof. We will prove that $|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p}$ i.e.

$$2^{-p}(x[p] - 1) < x_1 + x_2 + \ldots + x_n < 2^{-p}(x[p] + 1) \tag{1}$$

According to definition 2, we have

$$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m}|, n2^{-m} \tag{2}$$

We make $m = \lceil log_2 n + p + 1 \rceil$ such that

$$\frac{1}{2^m} \leq \frac{1}{n2^{p+1}} < \frac{1}{2^{m-1}} \tag{3}$$

According to (2) and (3), we get
$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m}| < n2^{-m} \leq 2^{-(p+1)}$ and

$$|x - (x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m}2^{-p}| < 2^{-(p+1)} \tag{4}$$

Then according to round's property, we have

$$|(x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m} - round((x_1[m] + x_2[m] + \ldots + x_n[m]) 2^{p-m})| \leq \frac{1}{2} \tag{5}$$

By multiplying 2^{-p}, Formula (5) can be changed into:

$$|(x_1[m] + x_2[m] + \ldots + x_n[m])2^{-m} - round((x_1[m] + x_2[m] + \ldots + x_n[m]) 2^{p-m})2^{-p})| \leq 2^{-(p+1)} \tag{6}$$

Then we can use $round((x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m})$ to replace $(x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m}$ in Formula (4), we can get

$$|x - round((x_1[m] + x_2[m] + \ldots + x_n[m])2^{p-m})2^{-p}| < 2^{-(p+1)} + 2^{-(p+1)} = 2^{-p} \tag{7}$$

i.e.

$$|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p} \tag{8}$$

In this addition algorithm, we distribute the same precision to every item. For example, if we want to get $x[p]$ whose expression is shown in Figure 1, we only need get $x_1[p+3]$, $x_2[p+3]$, $x_3[p+3]$ and $x_4[p+3]$ respectively. When n is a very large number this way can evidently reduce many items' precision required. Compared with the original algorithm, this way is more proper in terms of expression.

5 Algorithm with Precision Control

In section 4, we have solved the first problem that exits in constructive reals' computation. In this section we will consider solving the second problem. From what have been discussed, we can see that different operations have different complexity. A sin's operation is more complex than addition operation when is computed to up to the same precision. Our solution is to distribute different weight to single computation cell. The weight can control the precision that operation needs. First, we introduce a definition of single computation cell.

Definition 3. *(Single Computation Cell) A single computation cell is one of these cases: 1. A real number that interacts with other items only with addition or subtraction. 2. A result number that we get from operations except addition and subtraction and it interacts with other SCCs only with addition or subtraction.*

For example, when we want to deal with $x_1 + x_2 * x_3 + \sin x_4 + x_5 * x_6/x_7$, the number x_1, the result of $x_2 * x_3$, the result of $\sin x_4$ and the result of $x_5 * x_6/x_7$ are SCCs. We can simply put x_1's weight is 1, $x_2 * x_3$ is 4, $\sin x_4$ is 16 and

$x_5 * x_6/x_7$ is 8. The value of weight is based on the compute complexity of the SCC and is regulated to be the form of 2^k, $k \in N$, which make our computation convenient. We can compute single SCC use its original algorithm, but compute the whole expression use our algorithm. Now we will show an algorithm that can used to compute these expressions.

Algorithm 4. *If we want to compute* $x = x_1+x_2+\ldots+x_n$ *in order to meet some error range* 2^{-p}*, where* x_i*'s weight is* w_i *and every* x_i *is a SCC,* $i = 1, 2, \ldots, n$*. We have* $x[p] = round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + x_n[m - log_2 w_n]w_n 2^{p-m})$ *with* $m = \lceil -log_2(\frac{2^{-(p+1)}}{w_1+w_2+\ldots+w_n})\rceil$.

Proof. We will prove

$$|x - 2^{-p}x[p]| = |x_1 + x_2 + \ldots + x_n - 2^{-p}x[p]| < 2^{-p} \tag{1}$$

We make $m = \lceil -log_2(\frac{2^{-(p+1)}}{w_1+w_2+\ldots+w_n})\rceil$ such that

$$(w_1 + w_2 + \ldots + w_n)2^{-m} \leq 2^{-(p+1)} \tag{2}$$

According to definition 2, we have

$$\begin{aligned} &|x - (x_1[m - log_2 w_1]2^{log_2 w_1 - m} + x_2[m - log_2 w_2]2^{log_2 w_2 - m} + \ldots + x_n[m - \\ &log_2 w_n]2^{log_2 w_n - m})| = |x - (x_1[m - log_2 w_1]w_1 2^{-m} + x_2[m - log_2 w_2]w_2 2^{-m} \\ &+ \ldots + x_n[m - log_2 w_n]w_n 2^{-m})| < (w_1 + w_2 + \ldots + w_n)2^{-m} \leq 2^{-(p+1)} \end{aligned} \tag{3}$$

It is similar to the addition's proof above, we can get

$$\begin{aligned} &|(x_1[m - log_2 w_1]w_1 2^{-m} + x_2[m - log_2 w_2]w_2 2^{-m} + \ldots + x_n[m - log_2 w_n] \\ &w_n 2^{-m}) - round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + \\ &x_n[m - log_2 w_n]w_n 2^{p-m}) < 2^{-(p+1)} \end{aligned} \tag{4}$$

According to (3) and (4), we can get $|x - round(x_1[m - log_2 w_1]w_1 2^{p-m} + x_2[m - log_2 w_2]w_2 2^{p-m} + \ldots + x_n[m - log_2 w_n]w_n 2^{p-m})| < 2^{-(p+1)} + 2^{-(p+1)} = 2^{-p}$ that is to say $|x - x[p]| < 2^{-p}$.

For example, when we want to compute $x = x_1 + y_1 * y_2 + x_3 + x_4 + x_5$ that is $x = x_1 + x_2 + x_3 + x_4 + x_5$, where $x_2 = y_1 * y_2$ and $x_1, x_2, x_3, x_4, x_5$are SCCs. We can appoint x_1, x_3, x_4, x_5's weight to be 1 and x_2's weight to be 4. Then when we need to get $x[p]$, we should compute $x_1[p + 4]$, $x_2[p + 2]$, $x_3[p + 4]$, $x_4[p + 4]$, $x_5[p + 4]$ which is more efficient than compute $x_1[p + 4]$, $x_2[p + 4]$, $x_3[p + 4]$, $x_4[p + 4]$, $x_5[p + 4]$ that only use algorithm 3 to balance the precision. According to Algorithm 2 we can get $x_2[p + 2]$, and then we use Algorithm 4 to compute the whole expression.

6 Conclusions and Discussions

One of our aim is to avoid the unnecessary precision growth which has been talked in [17,18], in this paper we supply Algorithm 3 to balance every item in an expression including addition and subtraction. This way indeed improves exact arithmetic's speed. For example, we will do a computation that gets the sum of the harmonics series $\sum_{i=1}^{n} \frac{1}{i}$. The Table 1 below shows Algorithm 3's performance is much better than xrc1.1 which is an implementation based on V.Mnissier-Morain's Algorithm [3]. The software xrc1.1 can be gotten from [21].

Table 1. Computing $\sum_{i=1}^{n} \frac{1}{i}$

n	result'sprecision	xrc1.1(ms)	Algorithm 3 (ms)
1000	100	9.641	4.92
1000	1000	77.377	10.948
1000	10000	168.708	161.989
5000	100	141.016	58.564
5000	1000	187.183	106.209
5000	10000	455.335	364.182
10000	100	373.373	81.788
10000	1000	446.627	150.112
10000	10000	956.082	619.227

Table 2. Computing $\sum_{i=1}^{n}(\frac{1}{i*(i+1)} + \frac{1}{i})$

n	result'sprecision	xrc1.1(ms)	Algorithm 3 (ms)	Algorithm 4 (ms)
100	100	3.358	4.909	2.901
100	1000	11.811	10.911	10.256
1000	100	162.803	81.511	65.246
1000	1000	220.861	150.396	133.057
10000	100	19166.903	324.03	288.016
10000	1000	21070.736	900.51	894.279

But our way can only solve some of this problem. For example, $x_1 * x_2 * x_3$ also should have a way to make sure that every item has the similar precision, but because the result of multiplication's interval is too complex we failed to get an easy means just like we compute addition or subtraction. In the future, we may find a way to completely solve this problem.

Another aim is to take into account the operation's complexity which is first introduced by us. We control every item's precision according to its complexity. In this way, we can improve the performance of the exact arithmetic furthermore. For example, we will do a computation that gets the value of $\sum_{i=1}^{n}(\frac{1}{i*(i+1)} + \frac{1}{i})$. The Table 2 shows that Algorithm 4 has a better performance than xrc1.1 and Algorithm 3.

There is also a question in this algorithm that how large the weight should be. In our implementation, we can easily appoint it by ourselves according to test many times basic operations and we also can adjust it according to need.

Acknowledgements. The research was supported by Chinese 973 Program (2004CB719400), and the National Science Foundation of China (60403047, 60533070). The second author was supported by the project sponsored by a Foundation for the Author of National Excellent Doctoral Dissertation of PR China (200342), and a Program for New Century Excellent Talents in University (NCET-04-0088).

References

1. A. Edalat, R. Heckmann, Computing with real numbers: (i) LFT approach to real computation, (ii) Domain-theoretic model of computational geometry, Lecture Notes in Computer Science, vol.2395, Springer, 2002, pp.193-267.
2. J. Blanck, Efficient exact computation of iterated maps, The Journal of Logic and Algebraic Programming, vol.64, 2005, pp. 41-59.
3. V. Mnissier-Morain, Arbitrary precision real arithmetic: design and algorithms. The Journal of Logic and Algebraic Programming, Vol.64, 2005, pp. 13-19.
4. W. Gosper, Continued fraction arithmetics, Technical Report HAKMEM Item 101B, Artificial Intelligence Memo 239, MIT, 1972.
5. Jen Vuillemin, Exact real computer arithmetic with continued fractions, IEEE Transactions on Computers, Vol. 39, 1990, pp. 1087-1105.
6. E.Bishop, D.Bridges, Constructive Analysis, Springer-Verlag, 1985.
7. Hans-J. Boehm, Robert Cartwright, Mark riggle, and Michael J.O'Donnell, Exact real arithmetic:A case study in higher order programming, In Proceedings of the 1986 Lisp and Functional Programming Conference, 1986, pp. 162-173.
8. V. Lee, Optimizing Programs over the Constructive Reals, PhD thesis, Rice University, 1991.
9. Hans-J. Boehm The constructive reals as a Java library, The Journal of Logic and Algebraic Programming, vol.64, 2005, pp. 3-11.
10. P. Gowland, D. Lester, The correctness of an implementation of exact arithmetic, in: Proceedings of the Fourth Conference on Real Numbers and Computers, 2000.
11. D. Lester, P. Gowland, Using PVS to validate the algorithms of an exact arithmetic, Theoretical Computer Science, Vol. 291, 2003, pp. 203-218.
12. K. Briggs, Implementing exact real arithmetic in python, C++ and C, Theoretical Computer Science, vol.351, 2006, pp. 74-81.
13. P. Gowland, D. Lester A survey of exact arithmetic implementations, Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp. 30-47.
14. J. Blanck, Exact real arithmetic systems:Results of competition, Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp.389.
15. A. Edalat, P. J. Potts, A new representation for exact real numbers, Electronic Notes in Theoretical Computer Science, Vol.6, 1997, pp. 119-132.
16. P.J. Potts, Exact real arithmetic using M?bius transformations, PhD thesis, Imperial College, 1999.
17. Branimir Lambov, RealLib:An Efficient Implementation of Exact Real Arithmetic, http://www.bric.dk/ barnie/RealLib/, 2006.

18. Joris van der Hoeven, Computations with effective real numbers, Theoretical Computer Science, vol.351, 2006, pp. 52-60.
19. Norbert Th. Mller, The iRRAM: Exact Arithmetic in C++. Lecture Notes in Computer Science, vol.2064, Springer, 2001, pp. 222-252.
20. K. Weihrauch, An Introduction to Computable Analysis, Springer, 2000.
21. K. Briggs, xrc homepage, http://keithbriggs.info/xrc.html, 2005.