A Lazy Approach to Adaptive Exact Real Arithmetic using Floating-Point Operations

Ryan McCleeary*, Martin Brain†, Aaron Stump*
*Department of Computer Science, University of Iowa, Iowa City, USA
`ryan-mccleeary@uiowa.edu`, `aaron-stump@uiowa.edu`
† Department of Computer Science, University of Oxford, Oxford, UK
`martin.brain@cs.ox.ac.uk`

Abstract

Arithmetic operations with high degrees of precision are needed for an increasing number of applications.We propose an exact real arithmetic system that achieves adaptive precision using lazy infinite lists of floating-point values.

1 Introduction

There are an increasing range of applications where the precision of a double is not enough [1]. Two common ways to approach this problem are arbitrarily precise arithmetic and exact real arithmetic. Arbitrary precision requires the user to specify the precision up front and uses approximations of real numbers. MPFR does this by arbitrarily increasing the size of the significand [3]. Alternatively, Priest uses a finite list of floating-point numbers to approximate real values [5]. The value of his structures are the sum of the floating-point values. In order to do this he extensively uses an algorithm currently known as 2Sum. This function takes in two floating-point numbers and outputs two floating-point numbers. 2Sum only uses floating-point operations and satisfies the following properties on input (a, b) and output (s, t) (cf., Section 4.3 of [4]): $s + t = a + b$, s is the closest floating-point number to $a + b$, and $ulp(s) > |t|$. Because the precision must be fixed in advance, errors may accumulate beyond acceptable thresholds during sequences of computations, requiring the computations to be re-run with greater precision on input.

In contrast, exact real arithmetic allows the user to specify any degree of precision on output. For one example, Ciaffaglione and Gianantonia represent real numbers in $[-1, 1]$ with a lazily computed ternary stream of digits $a_1 : a_2 : a_3 : \cdots$ [2]. The stream is computed on demand to the required precision and its value is given by the function $[\,]_{str} : str \to \mathbb{R}$.

$$[a_1 : a_2 : a_3 : \cdots]_{str} = \Sigma_{i \in \mathbb{N}^+} a_i \cdot 2^{-1}$$

All numbers between -1 and 1 have an infinite number of representations. For example the number $\frac{2}{3}$ could be represented by the stream $1 : 0 : 1 : 0 : 1 \cdots$ or by the stream $1 : 1 : -1 : 1 : -1 \cdots$, because $0 : 1 = 0 \cdot 2^n + 1 \cdot 2^{n-1} = 1 \cdot 2^n - 1 \cdot 2^{n-1} = 1 : -1$. This representation allows for lazy left-to-right arithmetic.

We propose a new exact real arithmetic system built on floating-point operations. It uses lazy infinite lists of floating-point values, that satisfy similar invariants to Priest's finite lists [5], but makes use of left-to-right arithmetic like [2]. This gives us a system that can use fast floating-point operations, represents numbers sparsely, and computes with adaptive precision. Currently we have the following contributions: an isomorphism between real numbers and our representation of real numbers using an infinite sum of floating point values; algorithms for addition, negation, and multiplication; arguments for correctness on addition; and an implementation of the algorithms in the lazy functional programming language Haskell.

2 Exact Real Arithmetic with Streams of Floats

We use idealized floating-point numbers called blocks. A block consists of a vector of binary digits, a sign, and a mathematical integer for the exponent. The key difference between our representation and actual floating-point numbers is the mathematical integer for the exponent. Formally these are defined as follows:

Definition 2.1 ($block_k$)

$$blockVector_k = \{bv \in [0, k-1] \to \{0,1\} \,|\, bv[0] = 1 \lor \forall i \in [0, k-1], bv[i] = 0\}$$

$$block_k = \{(s, e, bv) | bv \in blockVector_k \;\&\; e \in \mathbb{Z} \;\&\; s \in \{-1, 1\}\}$$

The function $[\![\,]\!]_{block_k} : block_k \to \mathbb{Q}$ *gives us the rational number a block is representing*

$$[\![(s, e, bv)]\!]_{block_k} = s * 2^e * \textstyle\sum_{i=0}^{k-1} bv[i] * 2^{-i}$$

We define the set BCL_k to consist of finite and infinite lists of blocks, where we require zero overlap, as in [5]. Our system has adaptive precision as our basic operations are all done lazily. A BCL_k will not compute the next block unless it is necessary, but a BCL_k can compute an arbitrary number of blocks for any precision. We have defined in the lazy functional programming language Haskell addition, subtraction, and multiplication on BCL_k. Our addition algorithm uses 3 key functions: 2Sum, boothPrep, and zeroOverlap. 2Sum satisfies the specifications given in the introduction and zeroOverlap is equivalent to Priest's renormalization algorithm. The function boothPrep is what allows us to do lazy left-to-right computation by eagerly rounding up blocks e.g. boothPrep(1,0,[1,1,1,1]) = ((1,1,[1,0,0,0]) , (-1,-3,[1,0,0,0])).

```
add :: BCL -> BCL -> BCL
add [] gs = gs
add fs [] = fs
add (f:fs) (g:gs) = let (s,e) = (twoSum f g) in additionCoRec fs gs s (e:[])

additionCoRec :: BCL -> BCL -> Block -> BCL -> BCL
additionCoRec fs [] ZeroBlock [] = fs
additionCoRec [] gs ZeroBlock [] = gs
additionCoRec [] gs prev es = add gs (zeroOverlap (prev:es))
additionCoRec fs [] prev es = add fs (zeroOverlap (prev:es))
additionCoRec (f:fs) (g:gs) prev (e:es) =
  let (nextPrev1,error1) = twoSum f g in
   let(nextPrev2,error2) = twoSum nextPrev1 e in
    let(nextPrev3,newNeg) = boothPrep(nextPrev2) in
     let(output,nextPrev4) = twoSum prev nextPrev3 in
      let (nextPrev5:errors) = zeroOverlap(nextPrev4:error2:error1:newNeg:es) in
        output : (additionCoRec fs gs nextPrev5 errors)
```

In the future we plan to implement division, square root, complex functions, and transcendental numbers. We will also encode blocks into actual IEEE floating-point numbers using relative exponents to allow for an efficient implementation in C.

References

[1] D. Bailey. High-precision floating-point arithmetic in scientific computation. *Computing in Science and Engineering.*, 7(3):54–61, May 2005.

[2] A. Ciaffaglione and P. Gianantonio. A co-inductive approach to real numbers. In T. Coquand, P. Dybjer, B. Nordstrm, and J. Smith, editors, *Types for Proofs and Programs*, volume 1956 of *Lecture Notes in Computer Science*, pages 114–130. Springer Berlin Heidelberg, 2000.

[3] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007.

[4] JM. Muller, N. Brisebarre, F. de Dinechin, CP. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[5] D. Priest. Algorithms for arbitrary precision floating point arithmetic. In *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 132–145. IEEE Computer Society Press, 1991.