In Pursuit of Real Answers *

Angela Yun Zhu, Walid Taha, Robert Cartwright
Department of Computer Science, Rice University
Houston, TX 77005, USA
angela.zhu, taha, cork@rice.edu

Matthieu Martel
Laboratoire ELIAUS-DALI, Université de Perpignan
F-66860 Perpignan Cedex, France
matthieu.martel@univ-perp.fr

Jeremy G. Siek
Department of Electrical and Computer Engineering
University of Colorado at Boulder, USA
jeremy.siek@colorado.edu

Abstract

Digital computers permeate our physical world. This phenomenon creates a pressing need for tools that help us understand a priori *how digital computers can affect their physical environment. In principle, simulation can be a powerful tool for animating models of the world. Today, however, there is not a single simulation environment that comes with a guarantee that the results of the simulation are determined purely by a real-valued model and not by artifacts of the digitized implementation. As such, simulation with guaranteed fidelity does not yet exist.*

Towards addressing this problem, we offer an expository account of what is known about exact real arithmetic. We argue that this technology, which has roots that are over 200 years old, bears significant promise as offering exactly the right technology to build simulation environments with guaranteed fidelity. And while it has only been sparsely studied in this large span of time, there are reasons to believe that the time is right to accelerate research in this direction.

*This research was sponsored by the NSF under Award 0439017, 0720857, and 0747431. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, or the U.S. government.

1. Introduction

In the embedded systems community it is widely recognized that digital computers are permeating our physical world. Recently, there has been a growing consensus that this phenomenon of *cyber-physical systems* may require not only new tools but also new foundations to enable an effective understanding of such systems. In particular, there is a pressing need for methods that can provide us with useful, *a priori* accounts of how digital computers would affect a physical environment in which they are embedded.

In principle, simulation can serve as a powerful stride towards this goal. Unfortunately, no mainstream simulation environment available today comes with a clear, rigorous guarantee that would assure us that the results of the simulation are determined purely by a real-valued model and not by artifacts of the digitized implementation. Simulation tools with guaranteed fidelity do not yet exist.

The absence of such fidelity is a concern both in engineering and in science. When building safety critical systems, one would like to know that a behavior that appears possible in a simulation is indeed physically possible and not a result of numerical anomalies. In science, fidelity is absolutely essential for scientists to quickly determine whether an artifact in a physical simulation is a true phenomenon in the model or merely an artifact of the underlying implementation. Fidelity is also essential for the reproducibility of results.

For a simulation tool to provide fidelity guarantees it must be grounded in the mathematical study of real numbers, namely *real analysis*. This is, in essence, implied by the definition of what we want: To specify the correctness of an implementation, we need to use ideas from real analysis about approximations of increasing precision. Then, once we get into the details of the computation, and as we will get to see in some of the examples in this paper, we find that some fundamental truths about real-numbers (such as the undecidability of equality) cannot be avoided when we talk about high fidelity guarantees.

Exact real computation can be approached in several ways. A direct approach would be to compute explicitly with intervals representing precisely what we know about a certain real value. Such intervals can be represented, for example, by a pair of rationals. Then we can imagine all other operations being defined as working on intervals and producing answers that represent all possible answers for any given exact reals. We can call this method *interval arithmetic* [18, 20, 21, 1, 7].

While this approach can be effective for a wide range of applications, we believe that it can suffer from a particular type of computational inefficiency. Imagine a situation where we run a large computation and we get a result of unsatisfactory precision. What input do we need to provide in greater accuracy for the precision of the output to be improved? We expect that, in general, as we compose computation, the variance of needed precisions among inputs is likely to grow more dramatically, and as a result, only a very small number of inputs will really need to be provided in high-precision. This means that it becomes increasingly more wasteful to assume that all computation needs to be done in higher precision.

As such, we postulate that the most likely approaches to succeed in providing efficient real arithmetic computation will be based on lazy, explicit, and co-inductive ("infinite") representations of real numbers. Because this approach only computes to the precision needed, it avoids the problem with the direct approach.

Interestingly, it appears that the lazy approach has a long history going back at least to the work of Leslie and Cauchy in the 1800s [8]. At the same time, while exact real arithmetic may have seemed too expensive and too impractical in the past, a confluence of developments makes this an appropriate time to reconsider this judgment:

- Parallel computing resources: Multi-core, grid, and cloud computing systems all offer resources that can be employed effectively by highly parallelizable computations. Exact real arithmetic computations have the same purely functional properties as many other computations that are derived directly from mathematical computations. Furthermore, the granularity of individual operations in such a setting can be sufficiently large to take up non-trivial computing units.
- Feasibility of specialized hardware: FPGA, ASICS, and a wide range of technologies make it possible, today, to experiment with new architectural designs. At the same time, there is a growing consensus that it may be time to reconsider some very basic architectural aspects of microprocessor design. A better understanding of the needs of exact real arithmetic can provide critical guidance in this process.
- More than ever, a pressing need for accuracy: Today, even though some numerical libraries may have reasonable properties in terms of accuracy, it is rare that a library comes with any formal guarantees about accuracy. Even if that is the case, there are not methods for ensuring any properties when such libraries are composed (or even iterated!) For example, it is well known that the precision of transcendental functions can be very poor and vary greatly from system to system. Not only does this make it hard to have any basis for trusting the results of such computations, it makes it virtually impossible to use such computations to evaluate analytical models.

The question, then, is what are the right principle for designing lazy representation of exact real numbers?

1.1 Contributions

Towards addressing this problem, we offer an expository account of what is known about exact real arithmetic.

- We explain why the standard Base-2 representation for real numbers is not satisfactory, even for addition operation. We offer an intuitive explanation for why addition is not computable in this representation (Section 2).
- We explain two methods for solving this problem. The first method is to change the meaning of zero and one (Section 3). The second method is to add extra digits (Section 4).
- We show for the two representations how can $+$, $\times$ and $>$ be defined. We formalize theorems which state the soundness and effectiveness of the definitions.

Proofs of propositions and theorems can be found in an extended version of this paper [27].

2. Base-2 Representation Doesn't Work

A real number [1] in $[0, 1]$ has its integer part be 0 and every digit after the binary point be either 0 or 1. The traditional interpretation takes the n-th digit d after the binary

[1] We will focus on the fractional part of a real number and ignore its integer part. Real numbers and their operations with both integer part and fractional part can be treated by simple manipulation such as shifting.

point to mean $d \cdot 2^{-n}$. The value of a such number then is to add up the value of each digit. For example:

$$
\begin{array}{lll}
0.101(0)^{\omega} & \text{means} & 2^{-1} + 2^{-3} \\
0.01(1)^{\omega} & \text{means} & \sum_{i=-2}^{-\infty} 2^{i} = 2^{-1} \\
0.01(0)^{\omega} & \text{means} & 2^{-2} \\
0.01 \quad \text{or} \quad 0.01\ldots & \text{means} & [2^{-2}, 2^{-1}]
\end{array}
$$

Here, $(d_1 \ldots d_n)^{\omega}$ $(n \geq 1)$ denotes an infinite repetition of $d_1 \ldots d_n$, and an infinite sequence of unknown digits is denoted as $\ldots$ or ϵ (empty sequence). Different from floating point numbers, infinite sequence of digits does not have a least significant digit, and any operation must be implemented from left to right on the input sequences. However, this means even simple operations such as addition and subtraction are not computable. For example, suppose we wish to compute the result of adding two real numbers which start as follows:

$$0.0000000\ldots \qquad \text{and} \qquad 0.0111111\ldots$$

It is not clear whether the first digit of the result here should be a one or zero, since it depends on whether a carry will be generated later from the input sequences.

In the rest of this section, we will show why basic operations like addition cannot be defined with Base-2 representation, and define the formal semantics of this representation.

2.1. Expressible Intervals

In the traditional interpretation, for a finite sequence of digits A, the infinite sequence $0.A$ ranges from $0.A(0)^{\omega}$ to $0.A(1)^{\omega}$. For example, the sequence 0.0 means $[0, \frac{1}{2}]$, the sequence 0.1 means $[\frac{1}{2}, 1]$, and the sequence 0.00 means $[0, \frac{1}{4}]$ (Figure 1). When adding two numbers where only the first few digits of the inputs are known, we add the intervals corresponding to both inputs. The sum of the interval is guaranteed to include any possible results of the addition of input numbers. For example, we have $0.000 + 0.011 \in [\frac{3}{8}, \frac{5}{8}]$, $0.0000 + 0.0111 \in [\frac{7}{16}, \frac{9}{16}]$. The left most digits of the result can be inferred by finding a sequence of digits, where the interval it corresponded to must contain the sum of the intervals from the input.

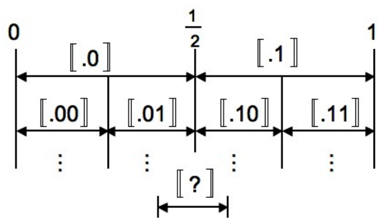


Figure 1. Base-2 Real Numbers

Recall the problem of computing the result of adding two real numbers which start as:

$$0.0000000\ldots \qquad \text{and} \qquad 0.0111111\ldots$$

If 0 keeps appearing in the first input and 1 keeps appearing in the second input, then after n pairs of 0 and 1, we would have the sum be lying in $[\frac{1}{2} - \frac{1}{2^{n+1}}, \frac{1}{2} + \frac{1}{2^{n+1}}]$. Now with the first digit be 0 ranges from 0 to $\frac{1}{2}$ and 1 ranges from $\frac{1}{2}$ to 1, the sum cannot have either 0 or 1 be its first digit. This makes the addition not computable. And the reason is because the intervals represented by 0 and 1 have only overlapped at one point. If there is a small overlap between the intervals represented by 0 and 1, a sufficient small interval should always be able to be included in one of the intervals.

2.2. Semantics

To better understand the Base-2 real representation and its operations, we give a formal definition for this representation and its interpretation. We also formally define the traditional addition operation and show that the operation is guaranteed to be correct but in many cases incomplete.

Definition 2.1 (Base-2 Real Numbers)**.**
Let $\mathbb{N}$ denote the set of naturals, and $\mathbb{Q}$ denote the set of rationals. Let $\mathbb{B} = \{0, 1\}$. Then a Base-2 real number *in $[0, 1]$ can be represented as an infinite stream of type $\mathbb{N} \rightarrow \mathbb{B}$.*

Definition 2.2 (Semantics of Base-2 Real Numbers)**.**
The semantics of any finite prefix of a real number is a function $[\![\_]\!] : (\exists n \in \mathbb{N}.\ \mathbb{B}^n) \rightarrow \mathbb{Q} \times \mathbb{Q}$ defined as follows:

$$
\begin{array}{lll}
[\![\epsilon]\!] & = & [0,\ 1] \\
[\![A0]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x,\ x + \frac{1}{2}(y - x)] \\
[\![A1]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x + \frac{1}{2}(y - x),\ y]
\end{array}
$$

We will use the following interval operations and relations throughout the paper:

Definition 2.3 (Basics for interval operations and relations)**.**

$$
\begin{array}{rcl}
[x_1, y_1] + [x_2, y_2] & = & [x_1 + x_2,\ y_1 + y_2] \\
[x_1, y_1] - [x_2, y_2] & = & [x_1 - y_2,\ y_1 - x_2] \\
[x, y]/z & = & [x/z,\ y/z] \\
[x_1, y_1] \subseteq [x_2, y_2] & \Leftrightarrow & x_1 \geq x_2 \ \text{ and } \ y_1 \leq y_2 \\
[x_1, y_1] < [x_2, y_2] & \Leftrightarrow & y_1 < x_2 \\
\big|[x, y]\big| \leq z & \Leftrightarrow & \max(|x|, |y|) \leq z
\end{array}
$$

where $|x|$ is the absolute value of x.

We can show that this interpretation converges to a real number as we look at a longer prefix:

Proposition 2.1. *For any $A \in \mathbb{B}^n$:*

- $[\![A]\!] \subseteq [0, 1]$

- $\left|[\![A]\!]\right| = (\frac{1}{2})^{|A|}$

where $|A|$ is the length of prefix A.

We now define the addition operation $\oplus$. Since we are considering real numbers in $[0, 1]$, we define $\oplus$ such that $[\![A \oplus B]\!] \simeq \frac{[\![A]\!]+[\![B]\!]}{2}$, where $A, B \in (\exists n \in \mathbb{N}.\ \mathbb{B}^n)$. The conventional addition is defined by left shifting the results.

Definition 2.4 (Addition of Base-2 Real Numbers.)**.**
For any digit $d \in \mathbb{B}$ and finite prefix $A, B \in (\exists n \in \mathbb{N}.\ \mathbb{B}^n)$:

$$\begin{array}{rclcl} A & \oplus & \epsilon & = & \epsilon \\ \epsilon & \oplus & B & = & \epsilon \\ dA & \oplus & dB & = & d(A \oplus B) \\ dA & \oplus & (1-d)B & = & \texttt{add\_one}(A \oplus B) \end{array}$$

$$\begin{array}{rcl} \texttt{add\_one}(\epsilon) & = & \epsilon \\ \texttt{add\_one}(1A) & = & 10A \\ \texttt{add\_one}(0A) & = & 01A \end{array}$$

Theorem 2.1 (Correctness of Addition)**.**
If $A \oplus B$ is defined, then $\frac{[\![A]\!]+[\![B]\!]}{2} \subseteq [\![A \oplus B]\!]$

Theorem 2.1 says that any digits produced from the addition operation is guaranteed to be correct. In other words, when more digits are available in the input sequences, there is no need to change the digits produced before.

Theorem 2.2. *Let $A = A_1 d A_2$ and $B = B_1 d B_2$, $|A_1| = |B_1| = l$. Then $|A \oplus B| \geq l + 1$.*

Theorem 2.2 says that addition on Base-2 real numbers can generate results of certain length if for some natural number n, the n-th digits in both input sequences are the same. Otherwise, the length of the result has no lower limit. The problem of deciding first digit in the result of $0.0000000\ldots + 0.01111111\ldots$ is one such example.

3. Proportional Relaxation

We now formalize binary numbers with proportional relaxation (Brouwer 1920, Turing 1937). The formalization has base 0 and 1. Digit 0 means zero and digit 1 means $\frac{1}{3}(\frac{2}{3})^{n-1}$ when it appears as the n-th digit after the binary point. The value of the real number is to add values of all the digits. Again, for a finite sequence of digits A, the infinite sequence $0.A$ ranges from $0.A(0)^\omega$ to $0.A(1)^\omega$. For example, the sequence 0.0 means $[0, \frac{2}{3}]$, the sequence 0.1 means $[\frac{1}{3}, 1]$, and the sequence 0.00 means $[0, \frac{4}{9}]$ (Figure 2). When adding two numbers where only their finite prefixes are known, we add the intervals corresponding to both inputs. The sum of the interval is guaranteed to include any possible results of the addition of input numbers. For example, we have $0.00 + 0.01 \in [0, \frac{4}{9}] + [\frac{2}{9}, \frac{6}{9}]$,

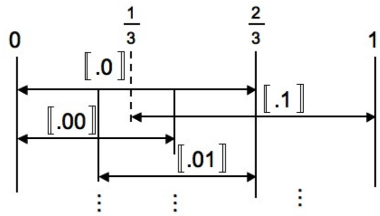


Figure 2. Binary Numbers with Proportional Relaxation

$0.000 + 0.011 \in [0, \frac{8}{27}] + [\frac{10}{27}, \frac{18}{27}]$. The latter one is contained in $[\frac{1}{3}, 1]$, so the result of $0.000+0.011$ must start with 0.1.

We formally define the representation described above as follows:

Definition 3.1 (Semantics of Binary Numbers with Proportional Relaxation)**.**
The semantics of any finite prefix of a real number is a function $[\![\_]\!] : (\exists n \in \mathbb{N}.\ \mathbb{B}^n) \to \mathbb{Q} \times \mathbb{Q}$ defined as follows:

$$\begin{array}{rcl} [\![\epsilon]\!] & = & [0,\ 1] \\ [\![A0]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x,\ x + \frac{2}{3}(y - x)] \\ [\![A1]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x + \frac{1}{3}(y - x), y] \end{array}$$

Again, this interpretation converges to a real number as we look at a longer prefix:

Proposition 3.1. *For any $A \in \mathbb{B}^n$:*

- $[\![A]\!] \subseteq [0, 1]$
- $\left|[\![A]\!]\right| = (\frac{2}{3})^{|A|}$

In the definition of addition of Base-2 real numbers, even the first digit of the result may not be decidable regardless of how many digits we known in the inputs. This is because some intervals cannot be expressed. If the interval stride over ranges represented by 0 and 1, no matter how short the interval is, no sequence of digits can represent the results.

In binary numbers with proportional relaxation, the intervals represented by 0 and 1 are overlapped with each other. The following lemma states the relationship between the size of an interval and the number of digits that can be decided for the interval. The relationship is guaranteed by the overlap in the semantics of digits 0 and 1.

Lemma 1. *For any rational interval $I = [x, y] \subseteq [0, 1]$, we can find a finite sequence A such that*

- $I \subseteq [\![A]\!]$
- $\frac{1}{3}|[\![A]\!]| < |I|$
- $|A| = \max\{0,\ \lfloor \log_{\frac{2}{3}} 3|I| \rfloor + 1\}$

Intuitively, Lemma 1 guarantees the computability of addition. Because when more digits are known from the inputs, the sum interval will have smaller size, which ensures that more digits are produced in the result. This is stated as the following theorem.

Theorem 3.1 (Computability of Addition)**.**
Let A and B be finite prefixes of two real numbers, and $|A|, |B| \geq n+4$. We can find a finite sequence C such that

- $[\![A]\!] + [\![B]\!] \subseteq [\![C]\!]$
- $|C| \geq n$

We now give a recursive definition of addition. The computation is conducted from left to right on the inputs. The carry is also generated from the most significant digit to the right. We define $\oplus$ such that $[\![A \oplus B]\!] \simeq (\frac{2}{3})^2([\![A]\!] + [\![B]\!])$, where $A, B \in (\exists n \in \mathbb{N}.\, \mathbb{B}^n)$. The actual summation of the inputs is derived by left shifting the results.

Definition 3.2 (Addition of Binary Numbers with Proportional Relaxation.)**.**
For any finite prefixes a_1a_2A and $b_1b_2B \in (\exists n \in \mathbb{N}.\, \mathbb{B}^{n+2})$, we have

$$
\begin{aligned}
& a_1a_2A \oplus b_1b_2B = \\
&\qquad \text{let} \quad s = \tfrac{1}{3}[(\tfrac{2}{3})^2(a_1+b_1) + (\tfrac{2}{3})^3(a_2+b_2)] \\
&\qquad \text{in} \quad \text{addc}\ (A, B, s) \\
& \text{addc}\ (aA, bB, r) = \\
&\qquad \text{let} \quad s = \tfrac{3}{2}r + \tfrac{1}{3}(\tfrac{2}{3})^3(a+b) \\
&\qquad \text{in} \quad \text{round}\ (s) :: \text{addc}\ (A, B, s - \tfrac{1}{2}\ \text{round}\ (s)) \\
& \text{addc}\ (A, \epsilon, r) = \epsilon \\
& \text{addc}\ (\epsilon, B, r) = \epsilon \\
& \text{round}\ (s) = \begin{cases} 0 & 0 \leq s < \frac{1}{2} \\ 1 & \frac{1}{2} \leq s \leq 1 \end{cases}
\end{aligned}
$$

where $d :: A$ means the concatenation of digit d and sequence A, ϵ means the empty sequence.

Theorem 3.2 (Correctness of Addition)**.** *For any $A, B \in (\exists n \in \mathbb{N}.\, \mathbb{B}^n)$, if $\min(|A|, |B|) = n+2$, $n \geq 0$ and $A \oplus B$ is defined, then:*

- $|A \oplus B| = n$
- $(\frac{2}{3})^2([\![A]\!] + [\![B]\!]) \subseteq [\![A \oplus B]\!]$

Theorem 3.3 (Computability of Multiplication)**.**
Let $A, B \in (\exists n \in \mathbb{N}.\, \mathbb{B}^n)$, and $|A| = |B| = n+4$. Then we can find a finite sequence C such that

- $[\![A]\!] \times [\![B]\!] \subseteq [\![C]\!]$
- $|C| \geq n$

Definition 3.3 (Multiplication of Binary Numbers with Proportional Relaxation.)**.** *For any finite prefixes $a_1a_2a_3A$ and $b_1b_2b_3B$, we have*

$$
\begin{aligned}
& a_1a_2a_3A \times b_1b_2b_3B = \\
&\qquad \text{let} \quad x = \tfrac{1}{3}[a_1 + \tfrac{2}{3}a_2 + (\tfrac{2}{3})^2 a_3] \quad \text{and} \\
&\qquad\qquad\ \ y = \tfrac{1}{3}[b_1 + \tfrac{2}{3}b_2 + (\tfrac{2}{3})^2 b_3] \quad \text{and} \\
&\qquad\qquad\ \ s = x * y \\
&\qquad \text{in} \quad \text{mult}\ (A, B, x, y, 4, s) \\
& \text{mult}\ (aA, bB, x, y, n, r) = \\
&\qquad \text{let} \quad s = \tfrac{3}{2}r + \tfrac{1}{3}(\tfrac{2}{3})^3(b * x + a * y + (\tfrac{2}{3})^n \tfrac{ab}{3}) \\
&\qquad \text{in} \quad \text{round}\ (s) :: \text{mult}\ (A, B, x + (\tfrac{2}{3})^n \tfrac{a}{3}, \\
&\qquad\qquad\ \ y + (\tfrac{2}{3})^n \tfrac{b}{3}, n+1, s - \tfrac{1}{2}\ \text{round}\ (s)) \\
& \text{mult}\ (A, \epsilon, c) = \epsilon \\
& \text{mult}\ (\epsilon, B, c) = \epsilon \\
& \text{round}\ (s) = \begin{cases} 0 & 0 \leq s < \frac{1}{2} \\ 1 & \frac{1}{2} \leq s \leq 1 \end{cases}
\end{aligned}
$$

Theorem 3.4 (Correctness of Multiplication)**.** *For any $A, B \in (\exists n \in \mathbb{N}.\, \mathbb{B}^n)$, if $\min(|A|, |B|) = n+4$, $n \geq 0$ and $A \times B$ is defined, then:*

- $|A \times B| = n$
- $[\![A]\!] \times [\![B]\!] \subseteq [\![A \times B]\!]$

Besides basic operations such as addition and multiplication, comparisons of two real numbers also has great importance in exact real arithmetic. For example, when conducting division, the first thing to do is to test whether the divisor is equal to 0. Comparison for real numbers is inherently semi-decidable, regardless of the way how it is defined. We give a definition for comparison of binary numbers with proportional relaxation. A direct implementation of this definition would either return a correct answer, or return an uncertainty of the comparison with an indication on how close the inputs are.

Definition 3.4 (Comparison)**.** *For any finite prefixes A and B, we have*

$$
\begin{aligned}
& A > B \quad \Leftrightarrow \quad \text{gt}\ (A, B, 0) \qquad \textit{where} \\
& \text{gt}\ (aA, bB, r) = \ \text{let} \quad s = \tfrac{3}{2}r + \tfrac{1}{2}(a-b) \quad \text{in} \\
&\qquad \begin{cases} \texttt{true} & \text{if} \quad s > 1 \\ \texttt{false} & \text{if} \quad s < -1 \\ \text{gt}\ (A, B, r) & \text{otherwise} \end{cases} \\
& \text{gt}\ (A, \epsilon, r) = \texttt{unknown} \\
& \text{gt}\ (\epsilon, B, s) = \texttt{unknown}
\end{aligned}
$$

Theorem 3.5.

$$
\begin{aligned}
& A > B \textit{ is } \texttt{true} && \Rightarrow \quad [\![A]\!] > [\![B]\!] \\
& A > B \textit{ is } \texttt{false} && \Rightarrow \quad [\![B]\!] > [\![A]\!] \\
& A > B \textit{ is } \texttt{unknown} && \Rightarrow \quad \big|[\![A]\!] - [\![B]\!]\big| \leq (\tfrac{2}{3})^{\min\{|A|,|B|\}}
\end{aligned}
$$

Other comparison operations like $<$, $\geq$, $\leq$ can be defined similarly.

4. Binary Numbers with Negative Digits

The design of binary numbers with proportional relaxation is inspired by the need of being able to express all intervals, which is achieved by providing redundancy on the semantics of different digits. Another way to introduce redundancy is to use an extra digit $\bar{1}$ besides zero and one (Leslie 1817, Cauchy 1840). The semantics of digit $\bar{1}$ is -1. Digit 0 means zero. Digit 1 and $\bar{1}$ means 2^{-n} and -2^{-n} respectively, when being placed as the n-th digit after the binary point. For a finite sequence of digits A, the infinite sequence $0.A$ would range from $0.A(\bar{1})^{\omega}$ to $0.A(1)^{\omega}$. So the sequence $0.\bar{1}$ means $[-1, 0]$, the sequence 0.0 means $[-\frac{1}{2}, \frac{1}{2}]$, and the sequence 0.1 means $[0, 1]$ (Figure 3).

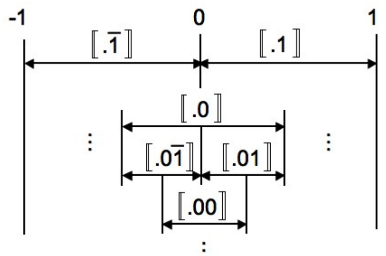


Figure 3. Binary Numbers with Negative Digits

We formally define this representation as follows:

Definition 4.1 (Binary Numbers with Negative Digits)**.**

Let $\mathbb{T} = \{\bar{1}, 0, 1\}$, then a real number in $[-1, 1]$ can be represented as an infinite stream of type $\mathbb{N} \to \mathbb{T}$.

Definition 4.2 (Semantics of Binary Numbers with Negative Digits)**.** *The semantics of any finite prefix of a real number is a function $[\![\_]\!] : (\exists n \in \mathbb{N}.\ \mathbb{T}^n) \to \mathbb{Q} \times \mathbb{Q}$ defined as follows:*

$$\begin{array}{lcl} [\![\epsilon]\!] & = & [-1,\ 1] \\ [\![A\bar{1}]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x,\ x + \frac{1}{2}(y-x)] \\ [\![A0]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x + \frac{1}{4}(y-x),\ x + \frac{3}{4}(y-x)] \\ [\![A1]\!] & = & \textit{let } [x, y] = [\![A]\!] \textit{ in } [x + \frac{1}{2}(y-x),\ y] \end{array}$$

For any real number, this interpretation converges to a real number as we look at a longer prefix:

Proposition 4.1.

- $[\![A]\!] \subseteq [-1, 1]$
- $\big|[\![A]\!]\big| = 2(\frac{1}{2})^{|A|}$

Computability of addition can be formalized in the same way as for binary numbers with proportional relaxation. We next define $\oplus$ such that $[\![A \oplus B]\!] \simeq \frac{[\![A]\!]+[\![B]\!]}{2}$, where $A, B \in (\exists n \in \mathbb{N}.\ \mathbb{B}^n)$. The actual summation of the inputs is derived by left shifting the results.

Definition 4.3 (Addition of Binary Numbers with Negative Digits)**.** *For any real numbers with prefixes aA and bB, $A, B \in (\exists n \in \mathbb{N}.\ \mathbb{B}^n)$, we have*

$$\begin{array}{l} aA \oplus bB = \text{ addc } (A, B, \dfrac{1}{2}(a+b)) \\ \text{addc } (aA, bB, r) = \\ \qquad \text{let} \quad s = 2r + \frac{1}{2}(a+b) \\ \qquad \text{in} \quad \text{round } (s) :: \text{ addc } (A, B, s - 2 \text{ round } (s)) \\ \text{addc } (A, \epsilon, r) = \epsilon \\ \text{addc } (\epsilon, B, r) = \epsilon \\ \text{round } (s) = \begin{cases} \bar{1} & -3 \le s < -1 \\ 0 & -1 \le s \le 1 \\ 1 & 1 < s \le 3 \end{cases} \end{array}$$

Theorem 4.1. *For any finite prefix A and B, if $\min(|A|, |B|) = n + 1$, $n \ge 1$ and $A \oplus B$ is defined, then:*

- $|A \oplus B| = n$
- $\frac{[\![A]\!]+[\![B]\!]}{2} \subseteq [\![A \oplus B]\!]$

We next define multiplication of binary numbers with negative digits in the following two definitions.

Definition 4.4 (Multiplication of a real number by a digit)**.**

$$\begin{array}{rcl} 1 \cdot dA & = & d(1 \cdot A) \\ 0 \cdot dA & = & 0(0 \cdot A) \\ \bar{1} \cdot dA & = & (0-d)(\bar{1} \cdot A) \end{array}$$

Definition 4.5 (Multiplication of two real numbers)**.** *Multiplication of two finite prefixes can be recursively defined as follows [23]:*

$$\begin{array}{rcl} a_1 a_2 A \times b_1 b_2 B & = & ((a_1 \cdot b_1 :: (b_2 \cdot A \oplus a_2 \cdot B)) \\ & & \oplus (a_1 \cdot b_1 :: a_2 \cdot b_1 :: a_2 \cdot b_2 :: A \times B)) \end{array}$$

Theorem 4.2. *For finite prefixes A and B, if $\min(|A|, |B|) = n + 2$, $n \ge 0$ and $A \times B$ is defined, then:*

- $|A \times B| = n$
- $([\![A]\!] \times [\![B]\!]) \subseteq [\![A \times B]\!]$

We have similar definition of comparison for binary numbers with negative digits as for binary numbers with proportional relaxation.

Definition 4.6 (Comparison)**.** *For any real numbers with finite prefix A and B, we have*

$$\begin{array}{l} A > B \ \Leftrightarrow\ \text{gt } (A, B, 0) \qquad \textit{where} \\ \text{gt } (aA, bB, r) = \text{ let } \ s = 2r + (a - b) \ \text{ in} \\ \qquad \begin{cases} \texttt{true} & \text{if } \ s > 1 \\ \texttt{false} & \text{if } \ s < -1 \\ \text{gt } (A, B, s) & \text{otherwise} \end{cases} \quad \text{gt } (A, \epsilon, r) = \texttt{unknown} \\ \text{gt } (\epsilon, B, s) = \texttt{unknown} \end{array}$$

The definition above satisfies:

Theorem 4.3.

$$\begin{array}{lcl} A > B \textit{ is } \texttt{true} & \Rightarrow & [\![A]\!] > [\![B]\!] \\ A > B \textit{ is } \texttt{false} & \Rightarrow & [\![B]\!] > [\![A]\!] \\ A > B \textit{ is } \texttt{unknown} & \Rightarrow & \big|[\![A]\!] - [\![B]\!]\big| \le 2 \cdot (\frac{1}{2})^{\min\{|A|,|B|\}} \end{array}$$

4.1 Related Work

Most existing computer systems approximate exact real numbers by floating point numbers [13]. The standard most commonly used is IEEE-754, which includes 32-bit single precision and 64-bit double precision representations [2]. Unfortunately, floating point arithmetic is inherently inaccurate. Only a very limited subset of real numbers may be represented exactly, and rounding errors occur in almost every floating point operation. Error analysis have been studied to overcome the problem, for example based on automatic differentiation [17], or using stochastic numbers to represent real numbers by tuples of randomly rounded floating-point numbers [25]. Some comparison between different methods based on a formal semantics for floating point numbers with errors can be found in [19].

A few academic tools exist with guaranteed fidelity in some very specific cases, such as systems with validated numerical solvers for Ordinary Differential Equations (ODE) in initial value problems [22, 9, 6, 3]. They are able to bound the distance between the computed and exact solution to ODEs. For example, Nedialkov et al [22] use interval-valued functions to approximate a solution to an initial value problem of ODEs by finding fixed points of some functions. Issues with interval arithmetic such as wrapping effects are addressed in detail.

Affine arithmetic improves the precision of interval arithmetic [10]. Compared to usual intervals, affine arithmetic makes it possible to reduce the over-approximation introduced by interval arithmetic by recording some relations between values. Recently, an abstract domain based on affine arithmetic has been proposed in [16]. Modal arithmetic is obtained by coupling an existential or universal quantifier to the usual intervals [12]. Modal intervals have been recently extended to generalized intervals which are intervals whose bounds are not constrained to be ordered [14]. Generalized intervals whose upper bound are less than the lower one are quantified existentially.

Different design and representations of exact real numbers using lazy representation has also been researched for a long time. Boehm, Cartwright, et al. [5] have implemented exact real arithmetic by representing real numbers as potentially infinite sequences of digits and evaluating on demand. They have compared this lazy implementation with a functional implementation and given some empirical comparison of the two techniques. Boehm and Cartwright provide more insights on the comparisons of different approaches to performing exact real arithmetic on a computer in [4].

Vuillemin [26] proposed a representation of computable real numbers by continued fractions and presented various incremental algorithms for basic arithmetic operations using the earlier work of Gosper [15], and for some transcendental function. Potts, Edalat and Escardó proposed a lazy representation of exact real numbers called *Linear Fractional Transformations* in [24]. They incorporated a representation of the non-negative extended real numbers based on the composition of linear fractional transformations with non-negative coefficients into the *Programming Language for Computable Functions (PCF)* with products. Later on, Edalat and Sunderhauf [11] use basic ingredients of an effective theory of continuous domains to spell out notions of computability of the reals and for functions on the real line.

While all of these works representing exact real numbers using lazy representation demonstrated impressive results, we believe that a better understanding of the design space of representations of exact real arithmetic may both yield a way to unify these approaches and accelerate advancement in this area.

5 Conclusions and Future Work

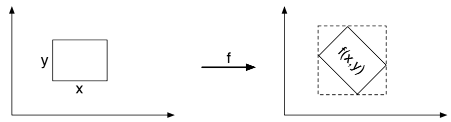
We began this paper by making a case for the importance and the timeliness of research on exact real computation, and followed this by an expository review of some of the basic issues that arise in exact real computation. To highlight some of the peculiarities of exact real computation, we discuss the difficulty in defining addition using the standard binary digit representation of fractions. We explain how this can be traced to the strictly hierarchical manner in which this representation forces intervals to be structured. This was followed by showing two different ways of solving this problem, namely, relaxing the meaning of digits and adding negative digits. For both these representations, we show how addition, multiplication, and comparison can be defined.

While space does not allow us in this paper, we believe that the issue of comparison deserves further discussion and analysis. Whereas here we have focused on pointing out these issues, our ultimate goal is to formalize these issues in a manner that allows us to classify and investigate the wide range of possible representations in a systematic and empirically justified manner. Because of their prominent role in scientific computing, in future work we hope to also identify the issues that arise in the introduction of the notions of division, exponentiation, continuous functions, integration, and differentiation.

Interval arithmetic suffers from the wrapping effect which makes the intervals grow artificially because of unavoidable over-approximations. For example, let us consider the function f which computes a rotation of angle θ:

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \end{pmatrix}$$

Intuitively, a pair (x, y) of intervals defines a box in the plane. The image $f(x, y)$ of this box by a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is not necessarily a box. However, in interval arithmetic, $f(x, y)$ must be approximated by a new pair of intervals which must strictly encompass the exact image. The dotted box on the right-hand-side of the figure below shows the most precise interval image of $f(x, y)$ with $\theta = \pi/4$.



Future work will explore ways of capturing and exploiting such dependence between different values.

References

[1] O. Aberth. *Introduction to Precise Numerical Methods, Second Edition*. Academic Press, Inc., Orlando, FL, USA, 2007.

[2] ANSI/IEEE. *IEEE Standard for Binary Floating-point Arithmetic*, std 754-1985 edition, 1985.

[3] AWA. http://www.math.uni-wuppertal.de/∼xsc/xsc/pxsc_software.html#awa.

[4] H. J. Boehm and R. Cartwright. Exact Real Arithmetic Formulating Real Numbers as Functions. *Research topics in functional programming*, pages 43–64, 1990.

[5] H. J. Boehm, R. Cartwright, M. Riggle, and M. J. OD́onnell. Exact Real Arithmetic: A Case Study in Higher Order Programming. In *LFP '86: Proceedings of the 1986 ACM conference on LISP and functional programming*, pages 162–173, New York, NY, USA, 1986. ACM.

[6] O. Bouissou and M. Martel. Grklib: a guaranteed runge kutta library. In *SCAN '06: Proceedings of the 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, page 8, Washington, DC, USA, 2006. IEEE Computer Society.

[7] H. Bronnimann, G. Melquiond, and S. Pion. The Design of the Boost Interval Arithmetic Library. *Theoretical Computer Science*, 351, 2006.

[8] A. Ciaffaglione and P. Di Gianantonio. A Certified, Corecursive Implementation of Exact Real Numbers. *Theoretical Computer Science*, 351(1):39–51, 2006.

[9] COSY. http://bt.pa.msu.edu/index_cosy.htm.

[10] L. H. De Figueiredo and G. Stolfi. Affine arithmetic: concepts and applications. *Numerical Algorithms*, 37, 2004.

[11] A. Edalat and P. Sunderhauf. A Domain-theoretic Approach to Real Number Computation. *Theoretical Computer Science*, 210:73–98, 1998.

[12] E. Gardenyes, H. Mielgo, and A. Trepat. Modal Intervals: Reason and Ground Semantics. In *Interval Mathematics*, number 212 in LNCS, pages 27–35. Springer-Verlag, 1985.

[13] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1), 1991.

[14] A. Goldsztejn and L. Jaulin. Inner and Outer Approximations of Existentially Quantified Equality Constraints. In *Twelfth International Conference on Principles and Practice of Constraint Programming*, LNCS. Springer-Verlag, 2006.

[15] W. Gosper. Continued fraction arithmetic, 1972. HAKMEN Item 101B, MIT Aritficial Intelligence Memo 239. MIT.

[16] E. Goubault and S. Putot. Under-approximations of Computations in Real Numbers based on Generalized Affine Arithmetic. In *Static Analysis Symposium*, number 4634 in LNCS. Springer-Verlag, 2007.

[17] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics. SIAM, 2000.

[18] M. Grimmer, K. Petras, and N. Revol. Multiple Precision Interval Packages: Comparing Different Approaches. In *Dagstuhl Seminar on Numerical Software with Result Verification*, number 2991 in LNCS. Springer-Verlag, 2003.

[19] M. Martel. An Overview of Semantics for the Validation of Numerical Programs. In *Verification, Model Checking and Abstract Interpretation*, volume 3385 of *LNCS*, pages 59–77. Springer-Verlag, 2005.

[20] R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, 1963.

[21] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. SIAM Press, 2009.

[22] N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.

[23] D. Plume. A calculator for exact real number computation, 1998. Available on line at http://www.cs.bham.ac.uk/ mhe/research.html.

[24] P. J. Potts, A. Edalat, and H. M. Escardó. Semantics of Exact Real Arithmetic. In *Procs of Logic in Computer Science*. IEEE Computer Society Press, 1997.

[25] J. Vignes. A Stochastic Arithmetic for Reliable Scientific Computation. *Mathematics and Computers in Simulation*, 35(3):233–261, 1993.

[26] J. E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers.*, 39(8):1087 – 1105, 1990.

[27] A.Y. Zhu, W. Taha, C. Cartwright, M. Martel, and J.G. Siek. In pursuit of real answers (extended version), 2009. Rice University, Technical report: TR09-01. Available on line at http://www.cs.rice.edu/∼taha/publications/preprints/2009-03-27-TR.pdf.