Using PVS to validate the algorithms of an exact arithmetic

David Lester*, Paul Gowland¹

Department of Computer Science, Manchester University, Oxford Road, Manchester M13 9PL, UK

Received 22 December 2000; received in revised form 4 June 2001; accepted 20 July 2001

Abstract

The whole point of exact arithmetic is to generate answers to numeric problems, within some user-specified error. An implementation of exact arithmetic is therefore of questionable value, if it cannot be shown that it is generating correct answers. In this paper, we show that the algorithms used in an exact real arithmetic are correct. A program using the functions defined in this paper has been implemented in ‘C’ (a HASKELL version of which we provide as an appendix), and we are now convinced of its correctness. The table presented at the end of the paper shows that performing these proofs found three logical errors which had not been discovered by testing. One of these errors was only detected when the theorems were validated with PVS. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Cauchy sequences; Computable reals; Correctness; Exact arithmetic; PVS

1. Introduction

The whole point of exact arithmetic is to generate answers to numeric problems, within some user-specified error. An implementation of exact arithmetic is therefore of questionable value, if it cannot be shown that it is generating correct answers. The implementation we describe is based on Cauchy sequences, a theory that is well established (see [9,12]). It is already known that the standard arithmetic operations are computable with respect to the Cauchy reals, but the Cauchy reals (without modification) cannot form the basis of an efficient implementation of the computable reals,

* Corresponding author.
E-mail addresses: dlester@cs.man.ac.uk (D. Lester), gowlandp@cs.man.ac.uk (P. Gowland).
¹ He was supported by an EPSRC Ph.D. Studentship.
because there is no consideration of the space needed to hold each rational approximation.

In this paper, our interest lies in minimizing the precision required for each argument to an operation whilst still ensuring that the answer for the operation is accurate.

2. The representation

At the heart of our implementation is the representation of a computable real number as fast binary Cauchy sequence. It is fast because we have an implicit modulus of convergence function which is the identity function; it is binary because the denominator of the \( p \)th element of the sequence is always \( 2^p \). This means that we do not need to store the denominator, and critically for space efficiency the size of the numerator grows linearly with the precision of the stored real number.

**Definition 2.1.** 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 has appeared in many papers [1,5,9]. Ref. [4] states it to be the most favourable definition for calculating the terms of a sequence to any accuracy desired.

As the main result of this section, we now show how our representation is related to the effective Cauchy representation [1,4,8,9,10,11,12] of the computable reals.

**Definition 2.2.** A computable real number \( x \) is represented as an effective Cauchy sequence if there is an infinite computable sequence of rationals \( \{n_0/d_0, n_1/d_1, \ldots, n_p/d_p, \ldots\} \), with \( d_i > 0 \), and a modulus of convergence function \( e: \mathbb{N} \to \mathbb{N} \) which is recursive, such that for all \( p \in \mathbb{N} \):

\[
k \geq e(p) \text{ implies } \left| x - \frac{n_k}{d_k} \right| < 2^{-p}.
\]

**Theorem 2.3.** Any computable real \( x \) represented as an effective Cauchy sequence \( (\hat{x}) \), with modulus of convergence function \( e \), can be converted into the fast binary Cauchy sequence \( (\tilde{x}) \) and vice versa. This interconversion being effective in both directions.

**Proof.** Let \( \phi \) and \( \psi \) be defined as

\[
\phi : \text{ECS} \to \text{FBCS},
\phi(\hat{x}) = \{ |\hat{x}[e(1)]|, |\hat{x}[e(2)]2^1|, \ldots, |\hat{x}[e(p + 1)]2^p|, \ldots \},
\]

\[
\psi : \text{FBCS} \to \text{ECS},
\psi(\tilde{x}) = \{ \tilde{x}[e(1)], \tilde{x}[e(2)]2^1, \ldots, \tilde{x}[e(p + 1)]2^p, \ldots \}.
\]

\[\text{We should probably be a bit more precise in insisting that } n_p \text{ is defined in a pair of recursive functions } \mathbb{N} \to \mathbb{N} \text{ of } p, \text{ one for the sign, and one for the magnitude of } n_p.\]
\( \psi : \text{FBCS} \rightarrow \text{ECS}, \)
\( \psi(\hat{x}) = \{\hat{x}[0], \hat{x}[1].2^{-1}, \ldots, \hat{x}[p].2^{-p}, \ldots\} \)

with modulus of convergence function: \( f(n) = n. \)

Notice that in all cases, calculating the \( p \)th element of the sequences is always effective. Showing that \( \psi(\hat{x}) \) is an effective Cauchy sequence is trivial, so we will show that \( \phi(\hat{x}) \) is a fast binary Cauchy sequence. Consider, for some \( p \in \mathbb{N}, \phi(\hat{x})[p] = [\hat{x}[e(p+1)].2^p] = m. \) We must show that

\[
\frac{m - 1}{2^p} < x < \frac{m + 1}{2^p}.
\]

We already know that, if \( \hat{x}[e(p+1)] = n/d \), then

\[
\frac{n}{d} - \frac{1}{2^{p+1}} < x < \frac{n}{d} + \frac{1}{2^{p+1}}.
\]

Assuming that \( d > 0 \), by lemma_A2 in Appendix A, we have that

\[
(2m - 1)d \leq n2^{p+1} < (2m + 1)d.
\]

This is enough to establish that

\[
\frac{m - 1}{2^p} \leq \frac{n}{d} - \frac{1}{2^{p+1}} \quad \text{and} \quad \frac{n}{d} + \frac{1}{2^{p+1}} \leq \frac{m + 1}{2^p}.
\]

Although this proof is straightforward, there is one important point: to convert a computable real represented as an effective Cauchy sequence we need one more bit of precision than we might have naively expected so that the rounding to a denominator that is a power of two does not lose accuracy.

3. The Cauchy reals

In our arithmetic package, we represent (computable) real numbers as Cauchy sequences. In PVS these sequences are most easily represented as functions \( c \), with the property that for all desired precisions \( p \) of the answer the (computable) real number \( x \) satisfies

\[
(c(p) - 1)2^{-p} < x < (c(p) + 1)2^{-p}.
\]

The precisions are taken to be natural numbers, and the function \( c \) returns integers. This is what is defined by the function cauchy_prop in theory cauchy. There is however a constraint on the acceptable functions of this form: the rational approximations \( c(p)/2^p \) must be converging to a particular real number \( x \). The predicate cauchy_real? captures this property. Finally, the type cauchy_real can be defined as the set of functions satisfying the predicate cauchy_real?, which happens to be non-empty, because \((\forall p:0)\)
represents the real number 0.

cauchy: THEORY
BEGIN

cauchy_prop(x: real, c: [nat → int]) : bool
= \forall (p: nat): c(p) - 1 < x × 2^p ∧ x × 2^p < c(p) + 1

cauchy_real?(c: [nat → int]): bool
= \exists (x: real): cauchy_prop(x, c)

cauchy_real: NONEMPTY_TYPE
= (cauchy_real?) CONTAINING (\lambda p: 0)

END cauchy

The elided part of the theory file defines subsets of the cauchy_reals to match the subsets of the reals in PVS—for example positive reals, non-negative reals etc.—and involves a great deal of repetition.

4. Correctness

We will now show that a number of algorithms for performing arithmetic on the cauchy_reals are correct. Ideally the algorithms should require the least possible accuracy of their arguments in order to ensure the required accuracy of their answers.

4.1. Integers

To provide some numbers for the system to work on, we provide a way to convert an integer into its cauchy_real representation.

int: THEORY
BEGIN

cauchy_int(n : int) : cauchy_real = (\lambda p: n × 2^p)

intLemma: LEMMA \forall (n : int) : cauchy_prop(n, cauchy_int(n))

END int

There appears to be some circularity here, in that one apparently needs to have proved intLemma in order to establish the type constraint of cauchy_int, but this circularity is illusory. We can prove the type constraint of cauchy_int without reference to intLemma:

\{1\} \forall(n : \text{int}) : \text{cauchy_real?}(\lambda p: n \times 2^p)

Expanding cauchy_real? we have

\{1\} \forall(n : \text{int}) : \exists(x : \text{real}) : \text{cauchy_prop}(x, \lambda p: n \times 2^p)
Skolemizing we have

\[
\{1\} \ \exists (x : \text{real}) : \text{cauchy\_prop}(x, \lambda p : n' \times 2^p)
\]

We can now instantiate \(x\) with \(n'\) which gives

\[
\{1\} \ \text{cauchy\_prop}(n', \lambda p : n' \times 2^p)
\]

Expanding \text{cauchy\_prop} is enough for PVS to spot that the theorem is true, as it would deduce

\[
\{1\} \ n' \times 2^p - 1 < n' \times 2^p \land n' \times 2^p < 1 + n' \times 2^p
\]

We may now proceed to prove \text{int\_lemma} in a nearly identical fashion.

4.2. Addition

To add two \text{cauchy\_reals}, we use \text{cauchy\_add}.

\text{add: THEORY}
\begin{verbatim}
BEGIN
  ...
  cauchy\_add(cx, cy: \text{cauchy\_real}): \text{cauchy\_real} =
  (\lambda p : \{(cx(p + 2) + cy(p + 2))/4\})

  add\_lemma: LEMMA \ \forall (x, y : \text{real}, cx, cy : \text{cauchy\_real}):
  \text{cauchy\_prop}(x, cx) \land \text{cauchy\_prop}(y, cy) \Rightarrow
  \text{cauchy\_prop}(x + y, \text{cauchy\_add}(cx, cy))
\end{verbatim}

\text{END add}

It is instructive to see how we persuade PVS of the validity of at least one part of our overall theory, so we display the results of proving the addition part.

After expanding the definition of \text{cauchy\_prop}, and then skolemizing and flattening, we have

\[
\begin{align*}
\{1\} \ & \forall (p: \text{nat}): cx'(p) - 1 < x' \times 2^p \land x' \times 2^p < cx'(p) + 1 \\
\{2\} \ & \forall (p: \text{nat}): cy'(p) - 1 < y' \times 2^p \land y' \times 2^p < cy'(p) + 1 \\
\{1\} \ & \text{cauchy\_add}(cx', cy')(p') - 1 < 2^{p'} \times (x' + y') \\
& \land 2^{p'} \times (x' + y') < \text{cauchy\_add}(cx', cy')(p') + 1
\end{align*}
\]

Instantiating the top quantifier in \{1\} and \{2\} with the terms: \(p' + 2\) gives

\[
\begin{align*}
\{1\} \ & cx'(p' + 2) - 1 < x' \times 2^{p' + 2} \land x' \times 2^{p' + 2} < cx'(p' + 2) + 1 \\
\{2\} \ & cy'(p' + 2) - 1 < y' \times 2^{p' + 2} \land y' \times 2^{p' + 2} < cy'(p' + 2) + 1 \\
\{1\} \ & [(cx'(p' + 2) + cy'(p' + 2))/4] - 1 < 2^{p'} \times (x' + y') \\
& \land 2^{p'} \times (x' + y') < [(cx'(p' + 2) + cy'(p' + 2))/4] + 1
\end{align*}
\]

Using the grind strategy from PVS then completes the proof of \text{add\_lemma}. 
To show that the algorithm cauchy_add uses no extraneous precision, consider \( cx(1) = 1 \) (which implies \( 0 < x < 1 \)) and \( cy(2) = 4 \) (which implies \( 0.75 < y < 1.25 \)). Then

\[
 cz(0) = \left\lfloor \frac{cx(1)}{2} + \frac{cy(2)}{4} \right\rfloor = \left\lfloor \frac{1.5}{2} \right\rfloor = 1
\]

with the implication that \( 0 < x + y < 2 \), when in fact the upper limit for \( x + y \) is 2.25.

### 4.3. Negation

Once more a trivial expansion suffices to show that the unary minus operation cauchy_neg is correct.

```plaintext
neg: THEORY
BEGIN
  cauchy_neg (cx : cauchy_real) : cauchy_real = (\( \lambda p : -cx(p) \))
  neg_lemma: LEMMA \( \forall (x : \text{real}, cx : \text{cauchy_real}) \):
    cauchy_prop(x; cx) \Rightarrow cauchy_prop(-x; cauchy_neg(cx))
END neg
```

### 4.4. Multiplication

With the multiplication algorithm, we come to the first of our serious proofs. The statement of the theorem, along with its associated lemmata comes to 89 lines, and the proof requires 1446 lines. Because of the size of the proofs, we have merely shown the proof of the main result, which in turn relies on Lemma mul_p6.

```plaintext
mul: THEORY
BEGIN
  \ldots
  x, y: VAR \text{real}
  n_1, n_2, m, r: VAR \text{int}
  cx, cy: VAR \text{cauchy_real}
  s_1, s_2, p: VAR \text{nat}
  \ldots
  mul_p6: LEMMA
    s_1 = \left\lfloor \log_2(|cx(0)| + 2) \right\rfloor + 3 \land s_2 = \left\lfloor \log_2(|cy(0)| + 2) \right\rfloor + 3 \land
    n_1 = cx(p + s_2) \land n_2 = cy(p + s_1) \land
    r = \left\lfloor \frac{(n_1 \times n_2)}{2^{p+s_1+s_2}} \right\rfloor \land
    cauchy_prop(x, cx) \land cauchy_prop(y, cy) \Rightarrow
    r - 1 < 2^p \times x \times y \land 2^p \times x \times y < r + 1
    \ldots
  cauchy_mul(c_1, c_2: \text{cauchy_real}): \text{cauchy_real} =
    (\lambda p: \text{LET s_1 = \left\lfloor \log_2(|cx(0)| + 2) \right\rfloor + 3, s_2 = \left\lfloor \log_2(|cy(0)| + 2) \right\rfloor + 3 \text{ IN } \left\lfloor \frac{(cx(p + s_2) \times cy(p + s_1))}{2^{p+s_1+s_2}} \right\rfloor})
  mul_lemma: LEMMA
```
cauchy_prop(x, cx) ∧ cauchy_prop(y, cy) ⇒
cauchy_prop(x × y, cauchy_mul(cx, cy))

END mul

Proof of mul_lemma. The algorithm for multiplication of two Cauchy reals is cauchy_mul; the key result mul_lemma follows from mul_p6.

In detail, after expanding cauchy_mul and the third occurrence of cauchy_prop, skolemizing, flattening and renaming some common subexpressions, we have

\[
\begin{align*}
\{ -1 \} \left( \frac{n_1' \times n_2'}{2^{p' + s_1' + s_2'}} \right) &= r' \\
\{ -2 \} cy'(p' + s_2') &= n_1' \\
\{ -3 \} cx'(p' + s_1') &= n_2' \\
\{ -4 \} \left\lceil \log_2(|cy'(0)| + 2) \right\rceil &= s_2' + 3 = s_2'' \\
\{ -5 \} \left\lceil \log_2(|cx'(0)| + 2) \right\rceil &= s_1' + 3 = s_1'' \\
\{ -6 \} cauchy_prop(x', cx') \\
\{ -7 \} cauchy_prop(y', cy')
\end{align*}
\]

We can then apply Lemma mul_p6 where s1 gets s1'', s2 gets s2'', cx gets cx'', cy gets cy'', x gets x'', y gets y'', r gets r'', n1 gets cx''(s_2'' + p''), n2 gets cy''(s_1'' + p''), and p gets p''. A simple assertion then establishes the result.

The precision required of the arguments to the multiplication algorithm is not as tight as it might be. For example, if we consider cz = cauchy_mul(cx, cy) with cx(0) = 0, then our algorithm suggests that to calculate cz(p) we will need to calculate cy(p + 4). Clearly, since the number represented by cx lies between −1 and 1, it would appear that we need in fact only evaluate cy(p).

There are three reasons for the extra precision in the definition of cauchy_mul.

1. The use of the floor instead of the ceiling operation. A slightly better result would be to use

\[
\text{LET } s_1 = \left\lceil \log_2(|cx(0)| + 1) \right\rceil + 3, \quad s_2 = \left\lceil \log_2(|cy(0)| + 1) \right\rceil + 3.
\]

The reason that this was not done was that there are more predefined lemmata in PVS dealing with floor than ceiling!

2. The additional +1 inside the log_2 terms cannot be avoided, because we only have approximations (cx(0), and cy(0)) from which to estimate the size of the intervals for both x and y.

3. Finally, the addition of 3 in s_1 and s_2 appears over generous, but was required to derive the result: mul_lemma. Although we may have been over generous, an attempt to prove mul_lemma, with +2 instead of +3 failed.

4.5. Inverse

Division is accomplished by using the invert function and multiplication. The type declarations have eliminated the possibility of division by 0, by restricting consideration to the non-zero reals.
The algorithm works by determining a suitable precision \( s \) that can guarantee that the Cauchy real is non-zero. From this, the inverted value can be calculated using \texttt{cauchy\_nz\_inv}.

\begin{verbatim}
inv: THEORY
BEGIN
  ...
p, s: VAR nat
r: VAR int
nzx: VAR nzreal
nzn: VAR nzint
minimum_inv (cx : cauchy\_nzreal) : nat = min\{s|3 \leq \|cx(s)\|\}
minimum_inv\_prop1: LEMMA(\forall nzx : cauchy\_nzreal, x : nzreal)
  s = minimum_inv(nzcx) \land cauchy\_prop(x, nzcx) \Rightarrow 2 \leq |x| \times 2^p
  ...
inv\_p: LEMMA 2 \leq |nzx| \times 2^p \land r = [2^{p+2} \times s^2 / nzn] \land
  nzn - 1 < nzx \times 2^{p+2} \times s^2 \land nzx \times 2^{p+2} \times s^2 < nzn + 1
  \Rightarrow (r - 1) < 2^n / nzx \land 2^n / nzx < (r + 1)
cauchy\_nz\_inv (cx : cauchy\_nzreal) : cauchy\_nzreal
  = (\lambda p : LET s = minimum_inv(cx) IN [2^{p+2s+2}/cx(p+2s+2)])
inv\_nz\_lemma: LEMMA \forall (x : nzreal, cx : cauchy\_nzreal):
  cauchy\_prop(x, cx) \Rightarrow cauchy\_prop(1/x, cauchy\_nz\_inv(cx))
END inv
\end{verbatim}

Proofs. Unfortunately, in our proof of Lemma \texttt{inv\_p} we run into a problem: we have to demonstrate separately the results for positive and negative arguments. An attempt to use the same theorems we have already proved for the negative values fails because our rounding operation is not symmetric under negation.

That is

\[-1.5 = -2 \neq -1.5 = 1\]

Notice that the problem is not solved by changing (\( \leq \)) to (\( < \)) everywhere; it is to do with the functional (or single-valued) nature of \texttt{cauchy\_nz\_inv}.

The proof of Lemma \texttt{inv\_lemma} falls into two cases: depending on whether the Cauchy real is representing zero or not. If it does represent zero, then the system will be able to deduce that \( x \) must be zero as well. In the second case we will apply Lemma \texttt{cauchy\_nz\_inv} to deduce our result.

The result of \texttt{inv\_nz\_lemma} is likely to be close to the best possible. However attempts to reduce the additive constant of +2 in \texttt{cauchy\_nz\_inv} to +1, led to failure, leading to a suspicion that the definition of \texttt{cauchy\_nz\_inv} might be the best possible without using an approximation to \( x \).
4.6. Square root

To prove this theorem correct, we first defined the square-root function in PVS, as this is not part of the standard system. An alternative would have been to have used the axiomatic formulation in the NASA Langley PVS library.

We have also restricted the domain of the function to the non-negative reals, in both the real and Cauchy real definitions.

```
sqrt: THEORY
BEGIN

::: sqrt

p1: LEMMA 0 < n ∧ r = ⌊√n⌋ ⇒ (r − 1)^2 ≤ n − 1 ∧ n + 1 ≤ (r + 1)^2

sqrt_p2: LEMMA
0 < n ∧ r = ⌊√n⌋ ∧ n − 1 < n×2^p ∧ n×2^p < n + 1
⇒ (r − 1)^2 < n×2^p ∧ n×2^p < (r + 1)^2

dauchy_nnsqrt(cx : dauchy_nreal) : dauchy_nreal

= (λ p : ⌊cx(2p)⌋)

sqrt_nn_lemma: LEMMA ∀(x : nreal, cx : dauchy_nreal) :
dauchy_prop(x, cx) ⇒ dauchy_prop(√x, dauchy_nnsqrt(cx))

END sqrt
```

**4.6.0.1 Proof.** Lemma_A1 along with the observation that \( r \geq 1 \) suffices to demonstrate sqrt_p1, and as a direct corollary, sqrt_p2 follows.

The need to calculate \( cx(2^p) \) to get an accurate result for \( cy(p) = ⌊\sqrt{cx(2^p)}⌋ \) will now be shown. Suppose that \( cx(2^p−1) = 0 \); then \( cy(p) = 0 \), but we would only know \( 0 < y < \sqrt{2}/2^p \), in other words the error is too large.

Notice that if the argument \( cx \) represents a number greater than 1, then we only need to calculate \( cx(p) \) to accurately calculate \( cy(p) \), which perhaps indicates that a revised accuracy (taking account of the approximate value of \( x \)) should be used.

4.7. Power

We have proved this result in order to prove properties about the power series we use in our implementation. To avoid having to give a value to \( 0^0 \) we have restricted the power \( n \) to be a positive natural number.

```
power: THEORY
BEGIN

::: power

cauchy_power_lt1(scx : dauchy_smallreal, n : posnat) : dauchy_real
= (λ p : ((scx( p + ⌊log2(n)⌋ + 3) × 2^{−(p+⌊log2(n)⌋+3)}) × 2^p))

power_lemma_l1: LEMMA dauchy_prop(scx, scx)
⇒ dauchy_prop(scx^p, dauchy_power_l1(scx, n))

cauchy_power(cx : dauchy_real, n : posnat) : dauchy_real
= (λ p : LET p_1 = p + ⌊log2(n)⌋ + 3 + n × (⌊log2(|cx(0)|+1)⌋+1) IN
```
Lemma \text{power lemma}_{\text{lt}1} proves our main result (\text{power lemma}) for “small” real values, \textit{i.e.} those with absolute value < 1. Much of the proof of \text{power lemma}_{\text{lt}1} is contained in \text{lemma A4}; the proof is by cases (many of the same ones used to prove \text{lemma A4}) and is largely using \text{lemma A4} in various creative ways. We have left discussion of \text{Lemma lemma A4} in Appendix A, as it is a property of real numbers rather than a connection between reals and Cauchy reals. To prove the general result \text{power lemma} we factor \(x\) as \(x = \lfloor \log_2(|x|+1) \rfloor \times y\) and \(|y|<1\), and then use various properties of the power function given in the PVS standard library.

### 4.8. Sum

We are able to sum cauchy reals in the following way:

```plaintext
cau\text{ch}_\text{sum}: \text{THEORY}
BEGIN
\text{cauchy}_\text{sum}: (\text{cx} : \text{nat} \rightarrow \text{cauchy real}, n : \text{nat}) : \text{cauchy real}
    = \left( \lambda p : \left( \frac{\sum_{i=0}^{m} \text{cx}_i (p + \lfloor \log_2(m+1) \rfloor + 2)}{2^{\lfloor \log_2(m+1) \rfloor + 2}} \right) \right)
\text{sum lemma}: \text{LEMM}\forall (m : \text{nat}) :
    (\forall (n : \text{nat}) : \text{cauchy prop}(x^n, \text{cx}_n)) \Rightarrow
    \text{cauchy prop} (\sum_{i=0}^{n} x_i, \text{cauchy sum}(\text{cx}, m))
END \text{sum}
```

The proof is a straightforward numeric induction on the length of the summation.

### 4.9. Power series

In this section we show how we can implement a power series using powers and sums.

```plaintext
\text{powerseries}: \text{THEORY}
BEGIN
\text{powerseries}(x : \text{real}, xs : \text{nat} \rightarrow \text{real}, n : \text{nat}) : \text{real}
    = \text{IF} \ n = 0 \ \text{THEN} \ x(0) \ \text{ELSE} \ x(0) + \sum_{i=1}^{n} x(i) \times x^i \ \text{ENDIF}
\text{cauchy_powerseries}(\text{cx} : \text{cauchy real}, \text{cxs} : \text{cau}\text{chys real}, n : \text{nat}) :
    \text{cauchy real}
    = \text{cauchy sum}(\text{\lambda i : IF} \ i = 0 \ \text{THEN} \ \text{cxs}_i \ \text{ELSE} \ \text{cauchy_mul(cxs}_i, \text{cauchy_power(cx, i)} \ \text{ENDIF}, n)
\text{powerseries lemma}: \text{LEMM}\forall (n : \text{nat}) : \text{cauchy prop}(x^n, \text{cxs}(n))) \land
    \text{cauchy prop}(x, \text{cx}) \Rightarrow
```
The proof is a simple composition of the proofs for powers and sums. The resulting algorithm is not particularly efficient, since it is repeatedly evaluating $x^i$ for $1 \leq i \leq m$.

5. Future work

There are a number of areas that could provide future work.

**Transcendental functions:** It would have been nice to have been able to make the direct connection between, for example, a sine function on reals ($\sin(x)$) and a proposed sine function on Cauchy reals (called $\text{cauchy\_sin}(cx)$, perhaps). One way to define $\sin(x)$ is to use the NASA Langley PVS real library to provide axiomatic definitions of the transcendental functions. We also have sufficient machinery to define the sine function on Cauchy reals as well:

$$\text{cauchy\_sin}(cx : \text{cauchy\_real}) : \text{cauchy\_real}$$

$$= \text{cauchy\_mul}(cx, \lambda p : \text{LET cx2 = cauchy\_mul}(cx, cx) \text{ terms = (}\lambda n : \text{cauchy\_mul}(\text{cauchy\_int}((-1)^n), \text{ cauchy\_inv}(\text{cauchy\_int}(\text{factorial}(n)))))) \text{ IN cauchy\_powerseries(cx2, terms, p)))}$$

Proving that this algorithm is correct would involve demonstrating that the Taylor series has sufficiently fast convergence.

**A non-axiomatic PVS reals library:** Alternatively, given that the current development is built on top of the standard PVS library without introducing any new axioms to the system,\(^3\) it might be fun to develop a non-axiomatic version of the NASA Langley library.

**Can the error bounds be reduced?** As we have discussed in the body of this paper, clearly in some cases they can. It is not so clear that verification of the resulting proofs will be very easy.

**A better power series function:** The algorithm for power series is not very good; ideally we should calculate the $x^{n+1}$ by multiplication of $x$ and $x^n$. Furthermore, a considerable economy can be obtained by having a sufficiently accurate approximation to $x$ and calculating all of the powers from that one approximation.

**Correctness of a calculator:** It would be useful to show that for any arithmetic expression, our exact arithmetic generated the correct answer when compared to

---

\(^3\) Readers should note that there are a few places where we have used non-constructive theorems from the PVS library, for example we have used the real\_complete theorem to define a square root function on the reals.
that generated by using traditional arithmetic. A start in this direction will be found in
the file congruence.pvs included in the PVS development available from

http://www.cs.man.ac.uk/arch/dlester/exact.html

6. Conclusion

Although the final result—that the representation and its associated operations are
correct—comes as no surprise, never the less, completion of this proof led to a number
of small changes to the implementation. All of these bugs had the ability to cause
arbitrarily large errors in certain (contrived) examples. For the morbid, we present the
changes in tabular form in Table 1.

The work most closely related to this paper is that of Valérie Méniessier–Morain
[3], in which she proved slightly more general results (in base b rather than just
base 2).

Our approach to implementing the transcendental functions also differs to [3] in that
in her work, several functions are implemented separately: e.g. \( \exp(x) \), \( \sin(x) \), etc. In
our work, we can instead define a power series algorithm and use this to construct
our transcendental functions. This should ease the difficult task of proving that our
algorithms are correct.

Müller’s iRRAM [6,7] can also be used to evaluate expressions to any accuracy.
However when performing calculations the iRRAM usually checks the error bounds
of the result, then if they have grown beyond \( 2^{-p} \) the result is thrown out and the
calculation repeated to a greater accuracy. One undocumented feature of the iRRAM
is that it is possible to prespecify the precision \( p \); if this is large enough then the
calculation is only performed once. This approach saves on space consumption as
intermediate values are not stored, however time consumption can be increased if there
is a need for recalculation of a result. A comparison of the approaches can be found
in [12].

The full PVS development can be found by following the links from

http://www.cs.man.ac.uk/arch/dlester/exact.html

<table>
<thead>
<tr>
<th>Function</th>
<th>Originally</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>cauchy_add</td>
<td>([\text{cx}(p + 1) + \text{cy}(p + 1)])</td>
<td>Testing</td>
</tr>
<tr>
<td>cauchy_mul</td>
<td>(s_1 = \lfloor \log_2(\text{cx}(0) + 1) \rfloor + 2); (s_2 = \lfloor \log_2(\text{cy}(0) + 1) \rfloor + 2)</td>
<td>Proof</td>
</tr>
<tr>
<td>cauchy_inv</td>
<td>(\text{LET } s = \min{s \mid 2 \leq</td>
<td>\text{cx}(s)</td>
</tr>
<tr>
<td>cauchy_power</td>
<td>(p_1 = p + \lfloor \log_2(n) \rfloor + 2 + n \times (\lfloor \log_2(\text{cx}(0) + 1) \rfloor + 1))</td>
<td>PVS</td>
</tr>
</tbody>
</table>
From the same page, one can also obtain two forms of the Haskell implementation, and (when the IRP issues are resolved) it is hoped that the ‘C’ version of the arithmetic package will be made available.

Appendix A. Useful real number lemmata

appendix: THEORY

BEGIN

lemma A1: \[ \forall (m, n : \mathbb{N}) : m \times m \leq n \land n < (m + 1) \times (m + 1) \equiv m = \lfloor \sqrt{n} \rfloor \]

lemma A2: \[ \forall (p, r : \mathbb{Z} \cap \mathbb{Q}) \ : q : \mathbb{N}^+ \) : \[ r = \lfloor p/q \rfloor \equiv (r - 1/2) \times q \leq p \land p < (r + 1/2) \times q \]

lemma A3: \[ \forall (n : \mathbb{N}) \ : q : \mathbb{N}^+ \) : \[ n = \lfloor \log_2(q) \rfloor \equiv 2^n \leq q \land q < 2^{n+1} \]

lemma A4: \[ \forall (n : \mathbb{N}) \ : x, e_1, e_2 : \mathbb{R} \) : \[ e_1 = 2^{-\lfloor \log_2(p) \rfloor + 2} \land e_2 = 2^{-n} \land -1 < x \land x < 1 \Rightarrow \]

\[ x^p - e_2 < (x - e_1)^p \land x^p - e_2 < (x + e_1)^p \land \]

\[ (x - e_1)^p < x^p + e_2 \land (x + e_1)^p < x^p + e_2 \]

END appendix

The last lemma (lemma A4) is stating that if we have an error of \( e_1 \) for \( x \) then the error for \( x^p \) will be less than \( e_2 \). The convoluted statement of the theorem captures the various different situations that arise. It is possible that \( (x + e_1)^p < (x - e_1)^p \), by choosing \( x = -0.1 \), and \( p = 2 \), for example.

Proof. The PVS proof of lemma A4 requires us to consider seven cases for \( x : x \in (-1, -e_1), x = -e_1, x \in (-e_1, 0), x = 0, x \in (0, e_1), x = e_1, x \in (e_1, 1) \). With negative values of \( x \) we need also to consider even and odd values of \( n \) separately. With some suitable generalizations, we can use induction to establish the result.

We have not provided full details of this proof for two reasons. First it is very long indeed (7170 lines of a total 21288 lines of PVS proof); secondly because we are not convinced that this proof could not be radically shortened. The key stage in the proof of lemma A4 is the use of induction to establish

lemma A4.5: \[ \forall (x, y : \mathbb{N}) \ : p_n : \mathbb{N}^+ \) : \[ -1 < x \land -1 < y \land (1 + y) \times p_n \times x < y \rightarrow (1 + x)^p_n < 1 + y \]

Appendix B. A Haskell implementation

The algorithms presented in the paper are incorporated into a Haskell implementation, pointers to which can be found at

http://www.cs.man.ac.uk/arch/dlester/exact.html
module Era where

data CR = CR_ (Int -> Integer)

-- The data type is a ‘wrapped’ function from precision to
-- numerator (denominator is 2^precision).

instance Num CR where
  (CR_ x') + (CR_ y') = CR_ (
p -> round_uk ((x'(p+2)+y'(p+2))%4))
  (CR_ x') * (CR_ y') = CR_ (
p -> round_uk ((x'(p+sy)*y'(p+sx))%
                       2^(p+sx+sy)))
  negate (CR_ x') = CR_ (
p -> negate (x' p))
  fromInteger n = CR_ (
p -> n*2^-p)

instance Fractional CR where
  recip (CR_ x')
    = CR_ (
p -> let s = head [n | n <- [0..], 3 <= abs (x' n)]
                  in round_uk (2^(-(2*p+2*s+2))*(x' (p+2*s+2))))

instance Floating CR where
  sqrt (CR_ x') = CR_ (
p -> floorsqrt (x' (2*p)))

power_CR :: CR -> Int -> CR
power_CR (CR_ x') n
  = if n == 0 then fromInteger 1
     else CR_ (
p -> let p' = p+sizeinbase (toInteger n) 2+3+n*s
                     in round_uk ((x' p'*(2^-p'))^n)*((2^-p)%1))
                     where s = 1 + sizeinbase (abs(x' 0)+1) 2

sum_CR :: [CR] -> CR
sum_CR xs
  = if n == 0 then fromInteger 0
     else CR_ (
p -> round_uk(sum [x' (p+s) |
                                  (CR_ x') <- xs]%(2^-s)))
                                  where n = toInteger (length xs)
                                         s = sizeinbase (n+1) 2 + 2

-- power_series takes as arguments:
--   a (rational) list of the coefficients of the power series
--   a function from accuracy to the number of terms needed
--   the argument x
power_series :: [Rational] -> (Int -> Int) -> CR -> CR
power_series ps terms x
  = CR_ (∀ p -> let (CR_ y') = f p in y' p)
  where f p = sum_CR 
    [fromRational (ps!!i) * power_CR x i | i <- [0..terms p]] in y' p)

-- As an example of the use of power_series, we give a domain
-- reduced sine function:

sin_dr :: CR -> CR
sin_dr x = x*power_series cs id (x*x)
  where cs = acc_seq (
        a n -> -a*(1%(2*n*(2*n+1))))

acc_seq :: (Rational -> Integer -> Rational) -> [Rational]
acc_seq f = scanl f (1%1) [1..]

-- GMP functions not provided by Haskell

sizeinbase :: Integer -> Int -> Int
sizeinbase n b = f (abs n)
  where f n = if n <= 1 then 1
              else 1 + f (n 'div' toInteger b)

floorsqrt :: Integer -> Integer
floorsqrt x = until satisfy improve x
  where improve y = floor ((y*y+x)%(2*y))
    satisfy y = y*y <= x && x <= (y+1)*(y+1)

round_uk :: Rational -> Integer
round_uk x = floor (x+1%2)

References