Math. Struct. in Comp. Science (2007), *vol.* 17, *pp.* 129–159. © 2007 Cambridge University Press
doi:10.1017/S0960129506005871 Printed in the United Kingdom

A monadic, functional implementation of real numbers

RUSSELL O'CONNOR

Institute for Computing and Information Science, Faculty of Science,
Radboud University Nijmegen
Email: r.oconnor@cs.ru.nl

Received 1 December 2005; revised 5 May 2006

Large scale real number computation is an essential ingredient in several modern mathematical proofs. Because such lengthy computations cannot be verified by hand, some mathematicians want to use software proof assistants to verify the correctness of these proofs. This paper develops a new implementation of the constructive real numbers and elementary functions for such proofs by using the monad properties of the completion operation on metric spaces. Bishop and Bridges's notion (Bishop and Bridges 1985) of regular sequences is generalised to what I call regular functions, which form the completion of any metric space. Using the monad operations, continuous functions on length spaces (which are a common subclass of metric spaces) are created by lifting continuous functions on the original space. A prototype Haskell implementation has been created. I believe that this approach yields a real number library that is reasonably efficient for computation, and still simple enough to verify its correctness easily.

1. Introduction

Several mathematical theorems rely on numerical computation of real number values in their proofs. One example is the disproof of the Mertens conjecture (Odlyzko and te Riele 1985). The Mertens conjecture claims that the absolute values of the partial sums of the Möbius function are bounded by the square root function. The conjecture implies the Riemann hypothesis. Odlyzko and te Riele's disproof involves computing the first hundred decimal digits of the first two thousand zeros of the Riemann zeta function.

Another example is Hales's famous proof of the Kepler conjecture (Hales 2002). This proof involves verifying thousands of non-linear inequalities over the real numbers. These inequalities were verified by special purpose software. Unfortunately, the peer review process has failed to accept fully that the proof is correct. After four years, the twelve referees could only claim to be 99% certain of the correctness of the proof. In response, Hales has created the *Flyspeck* project to create a software verified proof of the Kepler conjecture. A full software verified proof will verify both the mathematical and computational parts of the proof together.

Many other proofs rely on real number computation. All of these proofs require a software verified implementation of real number arithmetic before these proofs can be verified by computer. By implementation, I mean a software verified library of continuous

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

functions on $\mathbb{R}$ with which computations can be done to arbitrary precision. One such library has been created by Cruz-Filipe as a product of his software verified constructive proof of the fundamental theorem of calculus (Cruz-Filipe 2003).

When constructive mathematics is used for reasoning, algorithms are contained within the proofs. These algorithms can be executed in a functional language. One can see this as part of Bishop's program to view constructive mathematics as a programming language. Because Cruz-Filipe's proof is constructive, the functions in his library can actually be evaluated. However, Cruz-Filipe's construction was not designed with efficiency in mind, so, unfortunately, evaluation is not practical (Cruz-Filipe and Spitters 2003). Therefore, if one wants to be able to handle the problems encountered in the aforementioned mathematical proofs, a new library is needed.

This paper proposes a new constructive implementation of real numbers. An implementation that attempts to be simple and elegant, and at the same time practical enough to run the computations needed by the proof of the Kepler conjecture and other mathematical theorems. This work contributes not only to the *Flyspeck* project, but to any project requiring a verified library of functions for exact real arithmetic.

Section 2 introduces the notion of a regular function. Regular functions are a generalisation of Bishop and Bridges's notion of a regular sequence (Bishop and Bridges 1985). Section 3 discusses how regular functions, under a suitable equivalence relation, form a completion of any metric space.

Section 4 shows that the completion operation of a metric space forms a monad. A monad is an abstract structure consisting of a type constructor and three functions: unit, map and join. These three functions must satisfy seven monad laws, which are given in Section 4.1. Monadic programming techniques have been shown to be useful for programming tasks such as I/O, exceptions, non-determinism, and many more (Moggi 1991; Wadler 1992). The fact that the completion operation forms a monad is not all that exciting, but what is exciting is that these monad functions are very useful for defining functions over completed metric spaces by lifting uniformly continuous functions on the original metric space. Section 5 shows how this process can be used to define the real numbers and define the elementary functions over $\mathbb{R}$.

Because one goal of this work is to be able to evaluate expressions to a given precision in a reasonable amount of time, Section 6 gives some minor enhancements to the implementation to improve performance. Section 7 compares the run-time of a Haskell prototype implementation with the performance of other systems.

A second goal of this work is eventually to verify the proofs presented in this paper using a software proof assistant. To this end, I have included many theorems with detailed proofs in order to illustrate that verifying these proofs by software will be relatively easy. Those readers who are only interested in the data structures and operations may wish to skip over the details of the proofs.

I use constructive mathematics for all the proofs in this paper, so I remain agnostic as to whether or not functions are assumed to be computable. Of course, the goal is to make an implementation of the functions defined in this paper, so the computational interpretation is worth bearing in mind; however, both the constructive and classical interpretations are sound.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

1.1. *Notation*

The strictly positive rationals are denoted by $\mathbb{Q}^+$, while the non-negative rationals are denoted by $\mathbb{Q}^{0+}$; similar notation is used for reals and natural numbers. Equivalence relations are denoted by $\asymp$, while $=$ is reserved for intensional equality. I will be using sequences and functions to represent real numbers. It is possible for two different sequences or functions to represent the same real number, so it is important to distinguish between equality as functions and equivalence as real numbers.

The type of propositions is denoted by $\star$, and relations by functions to $\star$. If reasoning classically, $\star$ may be considered as a Boolean type.

In functional programming, functions are typically curried. Therefore, for a binary operator that would normally have the type $X \times X \Rightarrow X$, I will instead use the type $X \Rightarrow X \Rightarrow X$, though I will sometimes write function application as $f(x, y)$ for clarity.

As is also typical in functional programming, I will write anonymous functions using lambda expressions. For example, I will write the reciprocal function as $\lambda x.x^{-1}$.

2. Real numbers as regular functions of rationals

Abstractly, real numbers can be defined as any complete, Archimedean, ordered field, but in order to show that such a structure exists, it is necessary to produce a model of the real numbers. One common model of the real numbers consists of equivalence classes of Cauchy sequences of rational numbers.

Bishop and Bridges (1985) defines the real numbers as regular sequences of rational numbers with an equivalence relation. A sequence x_n is **regular** if

$$\forall nm : \mathbb{N}^+, |x_m - x_n| \leqslant m^{-1} + n^{-1},$$

and two regular sequences x and y are equivalent ($x \asymp y$) if

$$\forall n : \mathbb{N}^+, |x_n - y_n| \leqslant 2n^{-1}.$$

Regular sequences, under this equivalence relation, are isomorphic to Cauchy sequences. Every regular sequence is a Cauchy sequence, and every Cauchy sequence can be transformed into a regular sequence by using its modulus of convergence to select a suitable subsequence.

One can think of a regular sequence as a function that approximates a real number by taking a positive number n to a rational number x_n that is within n^{-1} of the real number that the sequence represents. Instead of using such coarse-grained approximations, one can generalise the concept of regular sequences to regular functions. A function $x : \mathbb{Q}^+ \Rightarrow \mathbb{Q}$ is **regular** if

$$\forall \varepsilon_1 \varepsilon_2, |x(\varepsilon_1) - x(\varepsilon_2)| \leqslant \varepsilon_1 + \varepsilon_2,$$

and two regular functions x and y are equivalent if

$$\forall \varepsilon, |x(\varepsilon) - y(\varepsilon)| \leqslant 2\varepsilon.$$

Regular functions are isomorphic to regular sequences. Any regular function can be transformed into a regular sequence by considering $\lambda n : \mathbb{N}^+.x(n^{-1})$, and any regular

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

sequence can be transformed into a regular function that maps ε to some x_n such that $n^{-1} \leqslant \varepsilon$.

One can think of a regular function as a function that approximates a real number by taking a positive rational number ε to a rational number that is within ε of the real number the function represents. Regular functions allow more fine-grained approximations than regular sequences allow. This fine granularity can prevent unnecessary over approximation when doing calculations. An example of this occurs in the implementation of multiplication in Section 5.

Regular functions can not only be used to construct the reals, but also to complete any metric space.

3. Metric spaces

A typical definition of a metric space is a space X with a distance function $d : X \Rightarrow X \Rightarrow \mathbb{R}^{0+} \cup \{\infty\}$ satisfying certain properties (Burago *et al.* 2001). However, the purpose of this paper is to define $\mathbb{R}$ as the completion of a metric space, so we require a definition of a metric space that does not presuppose the existence of $\mathbb{R}$. Given a traditional metric space, one can define a distance relation $B_\varepsilon(x, y) \stackrel{\text{def}}{=} d(x, y) \leqslant \varepsilon$ where $\varepsilon : \mathbb{Q}^+$. This distance relation does not depend on $\mathbb{R}$, but still completely characterises the metric space.

A tuple $(X, \asymp, B)$ is a **metric space**, where $\asymp$ is an equivalence relation, $B : \mathbb{Q}^+ \Rightarrow X \Rightarrow X \Rightarrow \star$ is a relation respecting the equivalence relation on X, and the following properties hold:

1 For all ε, B_ε is reflexive.
2 For all ε, B_ε is symmetric.
3 For all ε_1, ε_2, a, b, and c, if $B_{\varepsilon_1}(a, b)$ and $B_{\varepsilon_2}(b, c)$ hold, then $B_{\varepsilon_1+\varepsilon_2}(a, c)$ holds.
4 For all ε, a, and b, if $\forall \delta, B_{\varepsilon+\delta}(a, b)$ holds, then $B_\varepsilon(a, b)$ holds.
5 For all a and b, if $\forall \varepsilon, B_\varepsilon(a, b)$ holds, then $a \asymp b$.

Given a metric space, the traditional metric can be recovered (classically) by defining $d(a, b) \stackrel{\text{def}}{=} \inf\{\varepsilon | B_\varepsilon(a, b)\}$, where the infimum of the empty set is taken to be ∞. One can easily show that this definition of a metric space is equivalent to the traditional one.

The reason I use the symbol B for the distance relation is that if $B_\varepsilon(a)$ is considered as a predicate, the set of points b satisfying $B_\varepsilon(a)$ are exactly the points that are within ε of a, so $B_\varepsilon(a)$ can be thought of as a ball of radius ε around a.

Property 4 requires that $B_\varepsilon(a)$ be a closed ball. While the choice between open and closed balls is not very important from a classical perspective, there are some advantages to using closed balls when reasoning constructively. Because being closed is a negative statement, it does not contain any constructive information. One can avoid unnecessary computation and allow reasoning by contradiction by using negative statements wherever possible (Cruz-Filipe and Spitters 2003).

It is useful to note that if two points are within ε_1 of each other, they are also within ε_2 of each other whenever $\varepsilon_1 < \varepsilon_2$.

Lemma 1. For any ε, δ, a and b, if $B_\varepsilon(a, b)$ holds, then $B_{\varepsilon+\delta}(a, b)$ holds.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Proof. $B_\delta(b,b)$ holds by reflexivity. Therefore, $B_{\varepsilon+\delta}(a,b)$ holds. □

3.1. *Prelength spaces*

In a metric space, just because two points a and b are within l of each other does not mean that there exist curves of length approaching l that connect these points. A metric space that has such curves is known as a length space (Burago *et al.* 2001). Since the notion of curves presupposes the real numbers, I will introduce the weaker notion of a prelength space. A **prelength space** is a metric space such that

$$\forall ab\varepsilon\delta_1\delta_2, \varepsilon < \delta_1 + \delta_2 \Rightarrow B_\varepsilon(a,b) \Rightarrow \exists c, B_{\delta_1}(a,c) \wedge B_{\delta_2}(c,b).$$

This property implies that if two points a and b are within l of each other, then there is a path of points connecting a to b with consecutive points within ε of each other whose total length is less than $l + \varepsilon$.

Lemma 2. In a prelength space, for any a, b, ε, $\delta_0, \ldots, \delta_{n-1}$ such that $B_\varepsilon(a,b)$ holds and $\varepsilon < \delta_0 + \ldots + \delta_{n-1}$, there exist $c_0, \ldots, c_n$ such that $c_0 = a$, $c_n = b$, and $\forall i < n, B_{\delta_i}(c_i, c_{i+1})$ holds.

Proof. The proof is by induction on n. The case $n = 0$ is trivial. For the other cases, let $\gamma \stackrel{\text{def}}{=} \delta_0 + \ldots + \delta_{n-1} - \varepsilon$. There exists some c_{n-1} such that $B_{\delta_{n-1}}(c_{n-1}, b)$ and $B_{\varepsilon-\delta_{n-1}+\frac{\gamma}{2}}(a, c_{n-1})$. By induction, there exists $c_0, \ldots, c_{n-2}$ such that $\forall i < n-1, B_{\delta_i}(c_i, c_{i+1})$ because $\varepsilon - \delta_{n-1} + \frac{\gamma}{2} < \delta_0 + \ldots + \delta_{n-2}$. □

Prelength spaces are quite common; all the metric spaces considered in this paper are prelength spaces.

Theorem 1. $(\mathbb{Q}, =, B^{\mathbb{Q}})$ is a prelength space where $B^{\mathbb{Q}}_\varepsilon(a,b) \stackrel{\text{def}}{=} |a-b| \leqslant \varepsilon$.

Theorem 2. For any closed rational interval $[a,b]$, $([a,b], =, B^{\mathbb{Q}})$ is a prelength space.

3.2. *Completion of a metric space using regular functions*

One can define regular functions over any metric space X by using the distance relation. A function $x : \mathbb{Q}^+ \Rightarrow X$ is **regular** if

$$\forall \varepsilon_1 \varepsilon_2, B_{\varepsilon_1+\varepsilon_2}(x(\varepsilon_1), x(\varepsilon_2)),$$

and two regular functions x and y are equivalent ($x \asymp y$) if

$$\forall \varepsilon, B_{2\varepsilon}(x(\varepsilon), y(\varepsilon)).$$

Define the distance relation on regular functions as

$$B'_\varepsilon(x,y) \stackrel{\text{def}}{=} \forall \delta_1 \delta_2, B_{\varepsilon+\delta_1+\delta_2}(x(\delta_1), y(\delta_2)).$$

We call the type of regular functions over X the **completion** of X or $\mathfrak{C}(X)$. The following theorems show that the completion of a metric space is a metric space.

Theorem 3. If $(X, \asymp, B)$ is a metric space, then $(\mathfrak{C}(X), \asymp, B')$ is a metric space.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Proof.

1 Consider arbitrary δ_1 and δ_2. By Lemma 1, $B_{\varepsilon+\delta_1+\delta_2}(x(\delta_1), x(\delta_2))$ holds. Therefore, B'_ε is reflexive.
2 B' is symmetric because B is symmetric.
3 Suppose $B'_{\varepsilon_1}(x, y)$ and $B'_{\varepsilon_2}(y, z)$ hold. Let γ be arbitrary. Then

$$B_{\varepsilon_1+\delta_1+\frac{\gamma}{2}}(x(\delta_1), y(\frac{\gamma}{2}))$$

and

$$B_{\varepsilon_2+\frac{\gamma}{2}+\delta_2}(y(\frac{\gamma}{2}), z(\delta_2))$$

hold. Therefore,

$$B_{\varepsilon_1+\varepsilon_2+\delta_1+\delta_2+\gamma}(x(\delta_1), y(\delta_2))$$

holds. But since γ was arbitrary, $B'_{\varepsilon_1+\varepsilon_2}(x, y)$ holds.
4 Given ε, x, and y, suppose $\forall\delta_0, B'_{\varepsilon+\delta_0}(x, y)$ holds. Therefore,

$$\forall\delta_0\delta_1\delta_2, B_{\varepsilon+\delta_0+\delta_1+\delta_2}(x(\delta_1), y(\delta_2))$$

holds. But

$$\forall\delta_1\delta_2, B_{\varepsilon+\delta_1+\delta_2}(x(\delta_1), y(\delta_2))$$

holds, so $B'_\varepsilon(x, y)$ holds.
5 Given x and y, suppose $\forall\varepsilon, B'_\varepsilon(x, y)$ holds. For any ε and δ, $B_{\varepsilon+2\delta}(x(\delta), y(\delta))$ holds, so $B_{2\delta}(x(\delta), y(\delta))$ holds, and, therefore, $x \asymp y$. □

The following are useful lemmas about the distance function on regular functions.

Lemma 3. If $B_{\varepsilon_0}(x(\varepsilon_1), y(\varepsilon_2))$ holds, then $B'_{\varepsilon_0+\varepsilon_1+\varepsilon_2}(x, y)$.

Proof. Let δ_1 and δ_2 be arbitrary. $B_{\varepsilon_1+\delta_1}(x(\delta_1), x(\varepsilon_1))$ and $B_{\varepsilon_2+\delta_2}(y(\varepsilon_2), y(\delta_2))$ hold. Therefore, $B_{\varepsilon_0+\varepsilon_1+\varepsilon_2+\delta_1+\delta_2}(x(\delta_1), y(\delta_2))$ holds. □

Corollary 1. If $B_{\varepsilon_0}(x(\varepsilon_1), b)$ holds, then $B'_{\varepsilon_0+\varepsilon_1}(x, b')$ holds where b' is the regular function $b' \stackrel{\text{def}}{=} \lambda\varepsilon.b$.

Proof. By Lemma 3, $B'_{\varepsilon_0+\varepsilon_1+\varepsilon_2}(x, b')$ holds for every ε_2. Therefore, $B'_{\varepsilon_0+\varepsilon_1}(x, b')$ holds. □

The following theorem shows that the distance relation is equivalent to

$$\forall\delta, B_{\varepsilon+2\delta}(x(\delta), y(\delta)).$$

Theorem 4. For any ε, if $\forall\delta, B_{\varepsilon+2\delta}(x(\delta), y(\delta))$ holds, then $B'_\varepsilon(x, y)$ holds.

Proof. For any δ, $B_{\varepsilon+\frac{\delta}{2}}(x(\frac{\delta}{4}), y(\frac{\delta}{4}))$ holds. By Lemma 3, $B'_{\varepsilon+\delta}(x, y)$ holds. Since δ was arbitrary, $B'_\varepsilon(x, y)$ holds. □

The following theorem shows that the completion of a prelength space is a prelength space.

Theorem 5. If $(X, \asymp, B)$ is a prelength space, then $(\mathfrak{C}(X), \asymp, B')$ is a prelength space.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Proof. By Theorem 3, all that remains is to show that the prelength space property is preserved. Suppose $\varepsilon < \delta_1 + \delta_2$, and $B'_\varepsilon(x, y)$ holds. Let $\gamma \stackrel{\text{def}}{=} \frac{\delta_1+\delta_2-\varepsilon}{5}$. Then $B_{\varepsilon+2\gamma}(x(\gamma), y(\gamma))$ holds. There is some c such that $B_{\delta_1-\gamma}(x(\gamma), c)$ and $B_{\delta_2-\gamma}(c, y(\gamma))$ hold. So $B'_{\delta_1}(x, c')$ and $B'_{\delta_2}(c', y)$ holds by Corollary 1, where $c' \stackrel{\text{def}}{=} \lambda\varepsilon.c$. □

Typically, the completion of a space X is denoted by $\bar{X}$. Because I want to emphasise the monadic property of the completion, the Fraktur typeface $\mathfrak{C}$ is used in this paper.

3.3. *Uniform continuity*

Given two metric spaces $(X, \asymp, B^X)$ and $(Y, \asymp, B^Y)$, a function $f : X \Rightarrow Y$ is **uniformly continuous** with **modulus** $\mu : \mathbb{Q}^+ \Rightarrow \mathbb{Q}^+$ if

$$\forall \varepsilon x_1 x_2 : X.B^X_{\mu(\varepsilon)}(x_1, x_2) \Rightarrow B^Y_\varepsilon(f(x_1), f(x_2)).$$

A function $f : X \Rightarrow Y$ is **uniformly continuous** if there is some μ such that f is uniformly continuous with modulus μ. When a function f is uniformly continuous, I will write $f : X \to Y$ using a single-bar arrow. This indicates that f is not just a function, but it is a pair $(f, \mu_f) : (X \Rightarrow Y) \times (\mathbb{Q}^+ \Rightarrow \mathbb{Q}^+)$ consisting of a function and its modulus of continuity. This structure is a morphism in the category of metric spaces with uniformly continuous functions between them. When $f : X \to Y$, I will leave the projection function implicit and write $f(x)$ instead of $\pi_1(f)(x)$.

This is the usual definition of uniform continuity where the relation between ε and δ is explicitly given by μ.

Theorem 6. The identity function id is uniformly continuous with modulus id, and if $f : X \to Y$ and $g : Y \to Z$ are both uniformly continuous, then the composition $g \circ f : X \to Z$ is uniformly continuous with modulus $\mu_f \circ \mu_g$.

Proof. The proof is easy. □

3.4. *Remarks*

The definition of a metric space usually requires that the distance between two points is always finite. Because I will want to consider the space of all uniformly continuous functions between two metric spaces as a metric space in Section 4.2, I use the more liberal definition. It is worth noting that it is only the uniform space structure of a metric space that is important for the purposes of this paper. One can transform a metric space with infinite distances to one with only finite distances by defining a new metric $d'(a, b) \stackrel{\text{def}}{=} \min\{d(a, b), 1\}$. The uniform structure of the space does not change when using this new metric.

A prelength space is a weaker notion than a length space. For example, the rationals are not a length space because no two points can be connected by a continuous curve; however, the rationals are a prelength space. The name *prelength space* is chosen because of an analogy with precompact spaces. When a precompact space is completed it becomes a compact space; when a prelength space is completed it becomes a length space.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

An alternative definition for a regular function $x : \mathbb{Q}^+ \Rightarrow X$ would be

$$\forall \varepsilon_1 \varepsilon_2, B_{\max(\varepsilon_1,\varepsilon_2)}(x(\varepsilon_1), x(\varepsilon_2)).$$

This alternative definition would remove the need to restrict ourselves to prelength spaces (in particular, its use in Theorem 11). However, for efficiency reasons discussed in Section 6.1, I will stick with the definition based on Bishop and Bridges's regular sequences.

4. Completion is a monad

Any completion constructor forms a monad in the category of prelength spaces and uniformly continuous functions between them. The following defines the monad structure for the definition of completion given in Section 3.2. The unit : $X \to \mathfrak{C}(X)$ function is the obvious injection of X into $\mathfrak{C}(X)$:

$$\text{unit}(a) \stackrel{\text{def}}{=} \lambda \varepsilon.a.$$

Theorem 7. For any $a : X$, unit(a) is a regular function, and unit is uniformly continuous with modulus id.

The following two theorems show that the geometry of the space is preserved under this injection and that it is sound to think of $x(\varepsilon)$ as an approximation of x within ε.

Theorem 8. For any a, b and ε, we have $B_\varepsilon(a, b)$ holds if and only if $B'_\varepsilon(\text{unit}(a), \text{unit}(b))$ holds.

Theorem 9. For any ε, we have $B'_\varepsilon(x, \text{unit}(x(\varepsilon)))$ holds.

Proof. Let δ be arbitrary. $B_{\varepsilon+\delta}(x(\delta), x(\varepsilon))$ holds because x is regular, but this is the same as $B_{\varepsilon+\delta}(x(\delta), \text{unit}(x(\varepsilon))(\delta))$. By Theorem 4, $B'_\varepsilon(x, \text{unit}(x(\varepsilon)))$ holds. □

The function join : $\mathfrak{C}(\mathfrak{C}(X)) \to \mathfrak{C}(X)$ arises from the proof that completing a space twice is the same space as completing it once only:

$$\text{join}(x) \stackrel{\text{def}}{=} \lambda \varepsilon.x\left(\frac{\varepsilon}{2}\right)\left(\frac{\varepsilon}{2}\right).$$

Theorem 10. For any $x : \mathfrak{C}(\mathfrak{C}(X))$, join($x$) is a regular function, and join is uniformly continuous with modulus id.

Proof. Let ε_1 and ε_2 be arbitrary.

$$B_{\frac{\varepsilon_1}{2}+\frac{\varepsilon_2}{2}}\left(x\left(\frac{\varepsilon_1}{2}\right), x\left(\frac{\varepsilon_2}{2}\right)\right)$$

holds because x is a regular function.

By Theorem 9,

$$B_{\frac{\varepsilon_i}{2}}\left(x\left(\frac{\varepsilon_i}{2}\right), \text{unit}\left(x\left(\frac{\varepsilon_i}{2}\right)\left(\frac{\varepsilon_i}{2}\right)\right)\right)$$

hold.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Putting these together,

$$B_{\varepsilon_1+\varepsilon_2}(\text{unit}(\text{join}(x)(\varepsilon_1)), \text{unit}(\text{join}(x)(\varepsilon_2)))$$

holds.

By Theorem 8,

$$B_{\varepsilon_1+\varepsilon_2}(\text{join}(x)(\varepsilon_1), \text{join}(x)(\varepsilon_2))$$

holds, so $\text{join}(x)$ is a regular function.

Now let ε and δ be arbitrary, and suppose x_1 and x_2 are such that $B_\varepsilon(x_1, x_2)$ holds.

$$B_\delta(x_i, \text{unit}\left(\text{unit}\left(x_i\left(\frac{\delta}{2}\right)\left(\frac{\delta}{2}\right)\right)\right))$$

holds by two applications of Theorem 9. Therefore,

$$B_{\varepsilon+2\delta}(\text{unit}(\text{unit}(\text{join}(x_1)(\delta))), \text{unit}(\text{unit}(\text{join}(x_2)(\delta))))$$

holds. By two applications of Theorem 8,

$$B_{\varepsilon+2\delta}(\text{join}(x_1)(\delta), \text{join}(x_2)(\delta))$$

holds. By Theorem 4,

$$B_\varepsilon(\text{join}(x_1), \text{join}(x_2))$$

holds, as required. □

The function $\text{map} : (X \to Y) \Rightarrow (\mathfrak{C}(X) \to \mathfrak{C}(Y))$ lifts uniformly continuous functions on the base spaces to uniformly continuous functions on the completed spaces:

$$\text{map}(f) \stackrel{\text{def}}{=} \lambda x.f \circ x \circ \mu_f.$$

For metric spaces, $\text{map}(f)(x)$ is not always a regular function. Let X be the space $\{-1\} \cup \{n^{-1} | n : \mathbb{N}^+\}$ and Y be the space $\{-2\} \cup \{n^{-1} | n : \mathbb{N}^+\}$ with the induced metrics. The function $f : X \to Y$ sending -1 to -2 and n^{-1} to n^{-1} is uniformly continuous with modulus $\lambda\varepsilon.\min(\varepsilon, 1)$. Consider the regular function x that sends ε to -1 when $1 \leqslant \varepsilon$ and sends ε to some $n^{-1} \leqslant \varepsilon$ when $\varepsilon < 1$. In this case, $\text{map}(f)(x)$ is not a regular function.

Fortunately, $\text{map}(f)(x)$ is always a regular function when X is a prelength space.

Lemma 4. Given $f : X \to Y$ where X is a prelength space and Y is a metric space, consider arbitrary $a, b, \varepsilon_0, \ldots, \varepsilon_{n-1}$. Let $\delta_i \stackrel{\text{def}}{=} \mu_f(\varepsilon_i)$, $\delta \stackrel{\text{def}}{=} \delta_0 + \ldots + \delta_{n-1}$ and $\varepsilon \stackrel{\text{def}}{=} \varepsilon_0 + \ldots + \varepsilon_{n-1}$. If $B_\delta(a, b)$ holds, then $B_\varepsilon(f(a), f(b))$ holds.

Proof. Let ε_n be arbitrary and $\delta_n \stackrel{\text{def}}{=} \mu_f(\varepsilon_n)$. By Lemma 2, there are $c_0, \ldots, c_{n+1}$ such that $B_{\delta_i}(c_i, c_{i+1})$ hold, $a = c_0$ and $b = c_{n+1}$. By uniform continuity, $B_{\varepsilon_i}(f(c_i), f(c_{i+1}))$ hold. Thus $B_{\varepsilon+\varepsilon_n}(f(a), f(b))$ holds. Since ε_n was arbitrary, $B_\varepsilon(f(a), f(b))$ holds, as required. □

Theorem 11. If $f : X \to Y$ where X is a prelength space and Y is a metric space and $x : \mathfrak{C}(X)$, then $\text{map}(f)(x)$ is a regular function.

Proof. Let $y \stackrel{\text{def}}{=} \text{map}(f)(x)$. Consider arbitrary ε_1 and ε_2. Let $\delta_i \stackrel{\text{def}}{=} \mu_f(\varepsilon_i)$. Because x is a regular function, $B_{\delta_1+\delta_2}(x(\delta_1), x(\delta_2))$ holds. By Lemma 4, $B_{\varepsilon_1+\varepsilon_2}(f(x(\delta_1)), f(x(\delta_2)))$ holds, which is the same as $B_{\varepsilon_1+\varepsilon_2}(y(\varepsilon_1), y(\varepsilon_2))$. Therefore, y is a regular function. □

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Theorem 12. The function map(f) is uniformly continuous with modulus μ_f.

Proof. Consider arbitrary ε_0 and ε_1, and let $\delta_i \stackrel{\text{def}}{=} \mu_f(\varepsilon_i)$. Suppose $B_{\delta_1}(x_1, x_2)$ holds. So $B_{\delta_0+\delta_1+\delta_2}(x_1(\delta_1), x_2(\delta_2))$ holds. By Lemma 4, $B_{\varepsilon_0+\varepsilon_1+\varepsilon_2}(f(x_1(\delta_1)), f(x_2(\delta_2)))$ holds. Therefore, $B_{\varepsilon_0}(\text{map}(f)(x_1), \text{map}(f)(x_2))$ holds. □

The function bind : $(X \to \mathfrak{C}(Y)) \Rightarrow (\mathfrak{C}(X) \to \mathfrak{C}(Y))$ is defined in terms of join and map in the usual way:

$$\text{bind}(f) \stackrel{\text{def}}{=} \text{join} \circ \text{map}(f).$$

4.1. *Monad laws*

To prove that completion forms a monad, one needs to verify that the seven monad laws (Wadler 1992) hold.

Theorem 13. $\text{map}(\text{id}) \asymp \text{id}$, $\text{map}(f \circ g) \asymp \text{map}(f) \circ \text{map}(g)$ and $\text{map}(f) \circ \text{unit} \asymp \text{unit} \circ f$.

Proof. The proof is trivial. □

Theorem 14. $\text{map}(f) \circ \text{join} \asymp \text{join} \circ \text{map}(\text{map}(f))$.

Proof. Let x, ε_1 and ε_2 be arbitrary. Let $\delta_1 \stackrel{\text{def}}{=} \mu_f(\varepsilon_1)$, $\delta_2 \stackrel{\text{def}}{=} \mu_f(\frac{\varepsilon_2}{2})$, $a \stackrel{\text{def}}{=} x(\frac{\delta_1}{2})(\frac{\delta_1}{2})$ and $b \stackrel{\text{def}}{=} x(\delta_2)(\delta_2)$. By expanding the definitions, $\text{map}(f)(\text{join}(x))(\varepsilon_1) = f(a)$ and $\text{join}(\text{map}(\text{map}(f))(x))(\varepsilon_2) = f(b)$. Because x is a regular function, $B_{\frac{\delta_1}{2}+\delta_2}(x(\frac{\delta_1}{2}), x(\delta_2))$ and $B_{\delta_1+2\delta_2}(a, b)$ hold. By Lemma 4, $B_{\varepsilon_1+\varepsilon_2}(f(a), f(b))$ holds, as required. □

Theorem 15. $\text{join} \circ \text{unit} \asymp \text{id}$.

Proof. Let x, ε_1 and ε_2 be arbitrary. $\text{join}(\text{unit}(x))(\varepsilon_1) = x(\frac{\varepsilon_1}{2})$ and $\text{id}(x)(\varepsilon_2) = x(\varepsilon_2)$. Since x is a regular function, $B_{\varepsilon_1+\varepsilon_2}(x(\frac{\varepsilon_1}{2}), x(\varepsilon_2))$ holds, as required. □

Theorem 16. $\text{join} \circ \text{map}(\text{unit}) \asymp \text{id}$.

Proof. $\text{join}(\text{map}(\text{unit})(x))(\varepsilon) = \text{join}(\text{unit}(x))(\varepsilon)$, so the theorem follows from Theorem 15. □

Theorem 17. $\text{join} \circ \text{map}(\text{join}) \asymp \text{join} \circ \text{join}$.

Proof. By unfolding the definitions,

$$\text{join}(\text{map}(\text{join})(x))(\varepsilon_1) = x\left(\frac{\varepsilon_1}{2}\right)\left(\frac{\varepsilon_1}{4}\right)\left(\frac{\varepsilon_1}{4}\right)$$

and

$$\text{join}(\text{join}(x))(\varepsilon_2) = x\left(\frac{\varepsilon_2}{4}\right)\left(\frac{\varepsilon_2}{4}\right)\left(\frac{\varepsilon_2}{2}\right).$$

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Because x is a regular function,

$$B_{\frac{\varepsilon_1}{2}+\frac{\varepsilon_2}{4}}\left(x\left(\frac{\varepsilon_1}{2}\right), x\left(\frac{\varepsilon_2}{4}\right)\right),$$

$$B_{\frac{3\varepsilon_1}{4}+\frac{\varepsilon_2}{2}}\left(x\left(\frac{\varepsilon_1}{2}\right)\left(\frac{\varepsilon_1}{4}\right), x\left(\frac{\varepsilon_2}{4}\right)\left(\frac{\varepsilon_2}{4}\right)\right)$$

and

$$B_{\varepsilon_1+\varepsilon_2}\left(x\left(\frac{\varepsilon_1}{2}\right)\left(\frac{\varepsilon_1}{4}\right)\left(\frac{\varepsilon_1}{4}\right), x\left(\frac{\varepsilon_2}{4}\right)\left(\frac{\varepsilon_2}{4}\right)\left(\frac{\varepsilon_2}{2}\right)\right)$$

hold, and the last of these is what is required. □

4.2. *Lifting binary functions*

The function map can be used to lift unary functions from $X \to Y$ to $\mathfrak{C}(X) \to \mathfrak{C}(Y)$. However, one also wants to lift multi-argument functions such as binary functions. One can lift curried binary functions on base spaces to curried functions on completed spaces, but one first needs to consider the space of uniformly continuous functions as a metric space. This is done using the supremum norm.

Theorem 18. For any two metric spaces $(X, \asymp, B^X)$ and $(Y, \asymp, B^Y)$, the space of uniformly continuous functions from X to Y, $(X \to Y, \asymp, B^{X\to Y})$, is a metric space where we have $B_\varepsilon^{X\to Y}(f,g) \stackrel{\text{def}}{=} \forall a, B_\varepsilon^Y(f(a), g(a))$ and the equivalence relation $f \asymp g \stackrel{\text{def}}{=} \forall a, f(a) \asymp g(a)$.

Proof. All the properties of $B^{X\to Y}$ follow directly from the same properties for B^Y. □

Unfortunately, the space of uniformly continuous functions between two prelength spaces is not necessarily a prelength space. Consider the space of curves on the unit circle, $[0,1] \to S^1$, and consider two curves connecting two points where one goes clockwise and the other goes counter-clockwise. If $[0,1] \to S^1$ were a prelength space, then there would be a curve about halfway between these two curves, but there is no such curve.

Fortunately, one can still lift curried functions using map, because the proof that the result of map is a regular function (see Theorem 11) requires only that the *domain* be a prelength space. With curried functions (say $X \to (X \to X)$), it is the *range* that might not be a prelength space. The domain is still the prelength space X, so Theorem 11 still applies.

One can show that the monad is a strong monad by showing that map is uniformly continuous.

Lemma 5. Let X be a prelength space and Y be a metric space. Then map : $(X \to Y) \to (\mathfrak{C}(X) \to \mathfrak{C}(Y))$ is uniformly continuous with modulus id.

Proof. Let ε be arbitrary. Suppose $fg : X \to Y$ are such that $B_\varepsilon(f,g)$. Let ε_0 and $x : \mathfrak{C}(X)$ be arbitrary. Let $\delta_0 \stackrel{\text{def}}{=} \min(\mu_f(\varepsilon_0), \mu_g(\varepsilon_0))$ and $a_0 \stackrel{\text{def}}{=} x(\delta_0)$. Then

$$B_{\varepsilon_0}(\text{map}(f)(x), \text{map}(f)(\text{unit}(a_0)))$$

and

$$B_{\varepsilon_0}(\text{map}(g)(x), \text{map}(g)(\text{unit}(a_0)))$$

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

both hold by uniform continuity and Theorem 9. By Theorem 13,

$$\mathrm{map}(f)(\mathrm{unit}(a_0)) \asymp \mathrm{unit}(f(a_0))$$

and

$$\mathrm{map}(g)(\mathrm{unit}(a_0)) \asymp \mathrm{unit}(g(a_0)).$$

By Theorem 8,

$$B_\varepsilon(\mathrm{unit}(f(a_0)), \mathrm{unit}(g(a_0)))$$

holds. So putting these all together,

$$B_{\varepsilon+2\varepsilon_0}(\mathrm{map}(f)(x), \mathrm{map}(g)(x)).$$

Then, since ε_0 and x were arbitrary,

$$B_\varepsilon(\mathrm{map}(f), \mathrm{map}(g))$$

holds. □

With uniformly continuous function spaces in hand, one can construct the function $\mathrm{ap} : \mathfrak{C}(X \to Y) \to \mathfrak{C}(X) \to \mathfrak{C}(Y)$, which one can think of as a function that takes the limit of uniformly continuous functions:

$$\mathrm{ap}(f) \stackrel{\mathrm{def}}{=} \lambda x.\lambda\varepsilon.\mathrm{map}\left(f\left(\frac{\varepsilon}{2}\right)\right)(x)\left(\frac{\varepsilon}{2}\right).$$

Theorem 19. For any $f : \mathfrak{C}(X \to Y)$ and $x : \mathfrak{C}(X)$, we have $\mathrm{ap}(f)(x)$ is a regular function.

Proof. Let ε_1 and ε_2 be arbitrary, and let $f_i \stackrel{\mathrm{def}}{=} f(\frac{\varepsilon_i}{2}) : X \to Y$. Then, because f is regular, $B_{\frac{\varepsilon_1}{2}+\frac{\varepsilon_2}{2}}(f_1, f_2)$ holds. Let $y_i \stackrel{\mathrm{def}}{=} \mathrm{map}(f_i)(x)$. By Lemma 5, $B_{\frac{\varepsilon_1}{2}+\frac{\varepsilon_2}{2}}(y_1, y_2)$ holds. By Theorem 9, $B_{\frac{\varepsilon_1}{2}}(\mathrm{unit}(y_1(\frac{\varepsilon_1}{2})), y_1)$ and $B_{\frac{\varepsilon_2}{2}}(y_2, \mathrm{unit}(y_2(\frac{\varepsilon_2}{2})))$ hold. Putting these all together,

$$B_{\varepsilon_1+\varepsilon_2}\left(\mathrm{unit}\left(y_1\left(\frac{\varepsilon_1}{2}\right)\right), \mathrm{unit}\left(y_2\left(\frac{\varepsilon_2}{2}\right)\right)\right)$$

holds, and by Theorem 8,

$$B_{\varepsilon_1+\varepsilon_2}\left(y_1\left(\frac{\varepsilon_1}{2}\right), y_2\left(\frac{\varepsilon_2}{2}\right)\right)$$

holds, as required. □

Lemma 6. For any $f : \mathfrak{C}(X \to Y)$, $x : \mathfrak{C}(X)$ and ε, we have $B_\varepsilon(\mathrm{ap}(f)(x), \mathrm{map}(f(\varepsilon))(x))$ holds.

Proof. Let δ be arbitrary. Let $g \stackrel{\mathrm{def}}{=} f(\frac{\delta}{2})$ and $h \stackrel{\mathrm{def}}{=} f(\varepsilon)$. Because f is regular,

$$B_{\frac{\delta}{2}+\varepsilon}(\mathrm{map}(g)(x), \mathrm{map}(h)(x))$$

holds. By Theorem 9,

$$B_{\frac{\delta}{2}}\left(\mathrm{unit}\left(\mathrm{map}(g)(x)\left(\frac{\delta}{2}\right)\right),\ \mathrm{map}(g)(x)\right)$$

and

$$B_\delta(\mathrm{map}(h)(x), \mathrm{unit}(\mathrm{map}(h)(x)(\delta)))$$

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

hold. Putting these all together,

$$B_{\varepsilon+2\delta}\left(\mathrm{unit}\left(\mathrm{map}(g)(x)\left(\frac{\delta}{2}\right)\right),\ \mathrm{unit}(\mathrm{map}(h)(x)(\delta))\right)$$

holds, and, by Theorem 8,

$$B_{\varepsilon+2\delta}\left(\mathrm{map}(g)(x)\left(\frac{\delta}{2}\right),\ \mathrm{map}(h)(x)(\delta)\right)$$

holds. Then, by Theorem 4,

$$B_{\varepsilon}(\mathrm{map}(g)(x), \mathrm{map}(h)(x))$$

holds, as required. □

Theorem 20. For any $f : \mathfrak{C}(X \to Y)$, we have $\mathrm{ap}(f)$ is uniformly continuous with modulus $\lambda\varepsilon.\mu_{f(\frac{\varepsilon}{3})}(\frac{\varepsilon}{3})$.

Proof. Let ε be arbitrary and $\delta \stackrel{\mathrm{def}}{=} \mu_{f(\frac{\varepsilon}{3})}(\frac{\varepsilon}{3})$. Suppose that x_1 and x_2 are such that $B_\delta(x_1, x_2)$ holds. Then

$$B_{\frac{\varepsilon}{3}}\left(\mathrm{map}\left(f\left(\frac{\varepsilon}{3}\right)\right)(x_1),\ \mathrm{map}\left(f\left(\frac{\varepsilon}{3}\right)\right)(x_2)\right)$$

holds. Then, by Lemma 6,

$$B_{\frac{\varepsilon}{3}}\left(\mathrm{ap}(f)(x_i), \mathrm{map}\left(f\left(\frac{\varepsilon}{3}\right)\right)(x_i)\right)$$

hold. Together this means that $B_\varepsilon(\mathrm{ap}(f)(x_1), \mathrm{ap}(f)(x_2))$ holds, as required. □

Theorem 21. The function ap is uniformly continuous with modulus id.

Proof. Let ε be arbitrary. Suppose f_1 and f_2 are such that $B_\varepsilon(f_1, f_2)$ holds. Let x and δ be arbitrary. Because f_1 and f_2 are regular functions, $B_{\frac{\delta}{2}+\varepsilon}(f_1(\frac{\delta}{4}), f_2(\frac{\delta}{4}))$ holds. Therefore,

$$B_{\frac{\delta}{2}+\varepsilon}\left(\mathrm{map}\left(f_1\left(\frac{\delta}{4}\right)\right)(x), \mathrm{map}\left(f_2\left(\frac{\delta}{4}\right)\right)(x)\right)$$

holds. By Lemma 6,

$$B_{\frac{\delta}{4}}\left(\mathrm{ap}(f_i)(x), \mathrm{map}\left(f_i\left(\frac{\delta}{4}\right)\right)(x)\right)$$

holds. Putting these together yields

$$B_{\delta+\varepsilon}(\mathrm{ap}(f_1)(x), \mathrm{ap}(f_2)(x)),$$

and since δ and x were arbitrary,

$$B_{\varepsilon}(\mathrm{ap}(f_1), \mathrm{ap}(f_2))$$

holds, as required. □

By using ap, one can define a map2 function that lifts a curried function $f : X \to Y \to Z$:

$$\mathrm{map2}(f) \stackrel{\mathrm{def}}{=} \mathrm{ap} \circ \mathrm{map}(f) : \mathfrak{C}(X) \to \mathfrak{C}(Y) \to \mathfrak{C}(Z).$$

Other map functions can be made to lift n-ary functions by repeated use of ap.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

5. Functions of real numbers

Using this completion monad, it is now very easy to define the real numbers:

$$\mathbb{R} \stackrel{\text{def}}{=} \mathfrak{C}(\mathbb{Q}).$$

Functions operating on the real numbers can be created by using map or bind to lift uniformly continuous functions operating on the rationals. This is the advantage of using the monad functions. It is often easier to define a function on the rationals than on the reals because it is possible to decide for two rational numbers a and b which of $a < b$, $a = b$, or $a > b$ hold. The real numbers do not have this property.

Theorem 22.

— For all $a : \mathbb{Q}$, the functions $\lambda b. -b$ and $\lambda b.a + b$ are both uniformly continuous from $\mathbb{Q}$ to $\mathbb{Q}$ with moduli id.
— For all $a : \mathbb{Q}$, $\lambda b. \max(a, b)$ is uniformly continuous from $\mathbb{Q}$ to $[a, \infty[$ with modulus id.
— For all $a : \mathbb{Q}$, $\lambda b. \min(a, b)$ is uniformly continuous from $\mathbb{Q}$ to $]-\infty, a]$ with modulus id.
— The function $\lambda b.|b|$ is uniformly continuous from $\mathbb{Q}$ to $[0, \infty[$ with modulus id.
— For all $a : \mathbb{Q}$, $\lambda b.ab$ is uniformly continuous from $\mathbb{Q}$ to $\mathbb{Q}$ with modulus $\lambda \varepsilon.\varepsilon|a|^{-1}$ (or any modulus in the case when $a = 0$).

All these unary functions can be lifted, using map, to functions on $\mathbb{R}$. For the moment, one of the arguments to the binary functions $+$, $\cdot$, min and max must be rational. For example, for any $a : \mathbb{Q}$, one can use map to create the unary function $\lambda b.a + b : \mathbb{R} \to \mathbb{R}$, the function that translates by a rational number, but one does not yet have the binary function $\lambda ab.a + b : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$.

At this point, one can see why regular functions work better than regular sequences. For multiplication the modulus of continuity is $\lambda \varepsilon.\varepsilon|a|^{-1}$. If we had used regular sequences, we would need to find a point suitably far in the sequence. The result would be an over-approximation of the computation one needs. For a deep computation, these over-approximations would add up and can cause a lot of unnecessary work to be done. When we use regular functions, we can request exactly the amount of accuracy required.

5.1. *Reciprocal*

Not all of the functions one wishes to represent are uniformly continuous. One example of a non-uniformly continuous function is the reciprocal function, $\lambda x.x^{-1}$. Bishop and Bridges (1985) defines a continuous function as a function that is uniformly continuous on every closed interval in its domain. The general idea is for each x to find a domain containing x that the function is uniformly continuous on, and then evaluate this function at x.

Theorem 23. For every $a : \mathbb{Q}^+$, we have $\lambda x.x^{-1}$ is uniformly continuous on $[a, \infty[$ and $]\infty, -a]$, and both functions have a modulus of continuity of $\lambda \varepsilon.\varepsilon a^2$.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

One can now define x^{-1} for any $x : \mathbb{R}$ such that $x < 0$ or $0 < x$, where $0 < x \stackrel{\text{def}}{=} \exists \varepsilon.\varepsilon < x(\varepsilon)$, and similarly for $x < 0$. Suppose $0 < x$. Then there exists some $a : \mathbb{Q}$ such that $0 < a \leqslant x$. One can inject x into the domain $[a, \infty[$, and then invert it:

$$x^{-1} \stackrel{\text{def}}{=} \text{map}(\lambda b : [a, \infty[.b^{-1})(\max(a, x)).$$

Similarly, if $x < 0$, there exists some $a : \mathbb{Q}$ such that $x \leqslant a < 0$. In this case, we define the inverse as

$$x^{-1} \stackrel{\text{def}}{=} \text{map}(\lambda b :]-\infty, a].b^{-1})(\min(a, x)).$$

It is important to note that the result is independent of the choice of the domain that x is injected into. If $0 < a \leqslant x$ and $0 < a' \leqslant x$, then the result of x^{-1} is equivalent independent of whether x is injected into $[a, \infty[$ or x is injected into $[a', \infty[$.

Notice that the larger the value of a, the larger the modulus of continuity is. The larger the modulus of continuity is, the more efficient the lifted function is because less work is done in approximating the input. Therefore, it is helpful to find as large an a as is reasonable when computing the reciprocal.

5.2. *Binary functions of real numbers*

One can use map2 to lift addition to operate on $\mathbb{R}$.

Theorem 24. The function $\lambda a.\lambda b.a + b$ is uniformly continuous from $\mathbb{Q}$ to $\mathbb{Q} \to \mathbb{Q}$ with modulus id.

The definition of addition for $xy : \mathbb{R}$ is

$$x + y \stackrel{\text{def}}{=} \text{map2}(\lambda a.\lambda b.a + b)(x)(y).$$

The definition of multiplication on $\mathbb{R}$ is a little more complicated than the one just given for addition.

Theorem 25. For any $c : \mathbb{Q}^+$, the function $\lambda a.\lambda b.ab$ is uniformly continuous from $\mathbb{Q}$ to $[-c, c] \to \mathbb{Q}$ with modulus $\lambda \varepsilon.\varepsilon c^{-1}$.

The definition of multiplication for $xy : \mathbb{R}$ is

$$xy \stackrel{\text{def}}{=} \text{map2}(\lambda a.\lambda b.ab)(x)(\max(-c, \min(c, y))) \qquad \text{where } c \stackrel{\text{def}}{=} |y(1)| + 1.$$

Because multiplication is not uniformly continuous, some closed interval containing y must be found. This requirement is reflected in the fact that c is used in the definition of the modulus of continuity.

This definition of multiplication is asymmetric because only y is bound by a closed interval. The usual definition of multiplication in exact real arithmetic implementations finds closed intervals containing both x and y. The definition here is the natural definition of multiplication that one gets when one uses monad operations. It saves us from the unnecessary work of finding a closed interval for x.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

5.3. *Power series*

Other elementary functions can be defined in terms of power series. The limit of a convergent sequence can be found if the modulus of convergence is known. It is particularly easy to compute the limit of an alternating decreasing series.

Theorem 26. Suppose $\lim_{n\to\infty} a_n = 0$, a_n is alternating, and $\forall i, |a_{i+1}| < |a_i|$. Then $\sum_{i=0}^{\infty} a_i = x$ where $x(\varepsilon) \stackrel{\text{def}}{=} \sum_{\varepsilon<a_i} a_i$.

The power series of several elementary functions are alternating and decreasing on small rational inputs because the terms are bounded by a geometric series.

— For $-1 < a \leqslant 0$, $\exp^{\mathbb{Q}}(a) \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \frac{a^i}{i!}$.
— For $-1 < a < 1$, $\sin^{\mathbb{Q}}(a) \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} (-1)^i \frac{a^{2i+1}}{(2i+1)!}$.
— For $1 \leqslant a < 2$, $\ln^{\mathbb{Q}}(a) \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} (-1)^i \frac{(a-1)^{i+1}}{i+1}$.
— For $-1 < a < 1$, $\arctan^{\mathbb{Q}}(a) \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} (-1)^i \frac{a^{2i+1}}{2i+1}$.

These four functions can be extended to all rationals in their domains by repeated application of the following formulas to reduce the input to the above domains:

$$\begin{aligned}
\exp(a) &\asymp \exp^2\left(\frac{a}{2}\right) \\
\exp(a) &\asymp \exp^{-1}(-a) \\
\sin(a) &\asymp 3\sin\left(\frac{a}{3}\right) - 4\sin^3\left(\frac{a}{3}\right) \\
\ln(a) &\asymp \ln\left(\frac{a}{2}\right) + \ln(2) \\
\ln(a) &\asymp \ln\left(\frac{3}{4}a\right) + \ln\left(\frac{4}{3}\right) \\
\ln(a) &\asymp -\ln(a^{-1}) \\
\arctan(a) &\asymp \arctan(-a) \\
\arctan(a) &\asymp \frac{\pi}{2} - \arctan(a^{-1}) \qquad (\text{for } 0 < a) \\
\arctan(a) &\asymp \frac{\pi}{4} + \arctan\left(\frac{a-1}{a+1}\right) \qquad (\text{for } 0 < a).
\end{aligned}$$

The number π can be defined in terms of arctan in several ways. Below is a definition (Kurzweg 2001) that uses particularly small inputs to arctan.

$$\pi \stackrel{\text{def}}{=} (48\arctan(\tfrac{1}{38}) + 80\arctan(\tfrac{1}{57})) + (28\arctan(\tfrac{1}{239}) + 96\arctan(\tfrac{1}{268})).$$

Also, it is easy to define cos in terms of sin:

$$\cos(a) \stackrel{\text{def}}{=} 1 - 2\sin^2\left(\frac{a}{2}\right).$$

Theorem 27. The functions sin, cos and arctan are uniformly continuous on $\mathbb{Q}$ with moduli id.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Theorem 28. For every $a : \mathbb{Z}$, we have exp is uniformly continuous on $]-\infty, a]$ with modulus $\lambda\varepsilon.\varepsilon 2^{-a}$ if $a \leqslant 0$ and modulus $\lambda\varepsilon.\varepsilon 3^{-a}$ if $0 \leqslant a$.

Theorem 29. For every $a : \mathbb{Q}^+$, we have ln is uniformly continuous on $[a, \infty[$ with modulus $\lambda\varepsilon.\varepsilon a$.

These functions can all be lifted, using bind, to functions on $\mathbb{R}$, or whatever the completions of their domains are. The non-uniformly continuous functions exp and ln are defined in the same way as the reciprocal was defined; for each input x some suitable domain is found, and x is injected into that domain. Then the lifted function is evaluated.

Other elementary functions can be defined in terms of the four elementary functions given above:

$$\begin{aligned}
\tan(x) &\stackrel{\text{def}}{=} \frac{\sin(x)}{\cos(x)} \\
\sinh(x) &\stackrel{\text{def}}{=} \frac{\exp(x) - \exp(-x)}{2} \\
\cosh(x) &\stackrel{\text{def}}{=} \frac{\exp(x) + \exp(-x)}{2} \\
\tanh(x) &\stackrel{\text{def}}{=} \frac{\sinh(x)}{\cosh(x)} \\
x^y &\stackrel{\text{def}}{=} \exp(y \ln(x)) \\
\arcsin(x) &\stackrel{\text{def}}{=} \arctan\left(\frac{x}{\sqrt{1-x^2}}\right) \\
\arccos(x) &\stackrel{\text{def}}{=} \frac{\pi}{2} - \arcsin(x) \\
\operatorname{arcsinh}(x) &\stackrel{\text{def}}{=} \ln(x + \sqrt{1+x^2}) \\
\operatorname{arccosh}(x) &\stackrel{\text{def}}{=} \ln(x + \sqrt{x^2-1}) \\
\operatorname{arctanh}(x) &\stackrel{\text{def}}{=} \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right).
\end{aligned}$$

6. Improving efficiency

6.1. *Compression*

Calculations on rational numbers become more expensive as the size of the numerator and denominator become larger. One can keep rational numbers in reduced form by dividing through by the gcd of the numerator and denominator after each operation; however, for real numbers one can do better. Let us say that $\frac{n_1}{d_1}$ is simpler than $\frac{n_2}{d_2}$ if $|n_1| + |d_1| < |n_2| + |d_2|$. In any closed rational interval, there is a unique rational number that is simplest (Peyton Jones 2003). This idea can be used to **compress** real numbers; that is, to make the rational outputs of regular functions simple while keeping the real number it represents the same.

$$\operatorname{approx}(a, \varepsilon) \stackrel{\text{def}}{=} c \qquad \text{where } c \text{ is the simplest rational number in } [a-\varepsilon, a+\varepsilon].$$

$$\operatorname{compress}(x) \stackrel{\text{def}}{=} \lambda\varepsilon.\, \operatorname{approx}\left(x\left(\frac{\varepsilon}{2}\right),\ \frac{\varepsilon}{2}\right).$$

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Theorem 30. For any $x : \mathbb{R}$, compress$(x) \asymp x$.

Proof. Let ε be arbitrary. Then

$$B_{\frac{\varepsilon}{2}}\left(\text{compress}(x)(\varepsilon), x\left(\frac{\varepsilon}{2}\right)\right)$$

holds by the definition of approx. Because x is a regular function,

$$B_{\frac{3\varepsilon}{2}}\left(x\left(\frac{\varepsilon}{2}\right),\ x(\varepsilon)\right)$$

holds, so

$$B_{\varepsilon}(\text{compress}(x)(\varepsilon), x(\varepsilon))$$

holds, as required. □

Generous use of compress can greatly increase the efficiency of the computation. In my implementation, every function operating on $\mathbb{R}$ passes its input through compress first.

Note that if one uses the alternative defintion of regular function given in Section 3.4, this definition of compress does not work. I do not see a way of defining a version of compress for the alternative definition of regular function, and without compress, the rational numbers used in computation become too large for practical evaluation.

6.2. *Newton approximation*

Although one can define $\sqrt{x}$ as $\exp(\frac{1}{2}\ln(x))$, the square root function is common enough to warrant its own, more efficient, definition. If $a : \mathbb{Q}$ is such that $1 \leqslant a < 4$, one can compute $\sqrt{a}$ by taking the limit of Newton approximations. Let the first approximation be $b_0 \stackrel{\text{def}}{=} \frac{a+1}{2}$, and let successive approximations $b_{i+1} \stackrel{\text{def}}{=} \text{approx}_{\mathbb{Q}}(\frac{a+b_i^2}{2b_i}, 2^{-2^{i+1}-1})$. The first approximation has an error of at most $\frac{1}{2}$. Each successive approximation squares the error.

Theorem 31. For any $a : \mathbb{Q}$, such that $1 \leqslant a < 4$, let $x(\varepsilon) \stackrel{\text{def}}{=} b_n$, where n is the first natural number such that $2^{-2^n} \leqslant \varepsilon$. Then x is a regular function.

Proof. It suffices to show that for all n, $\sqrt{a} - 2^{-2^n} \leqslant b_n \leqslant \sqrt{a} + 2^{-2^n}$ and $1 \leqslant b_n$.

Suppose $n = 0$. For the bound below, note that $0 \leqslant \frac{(a-1)^2}{4} = b_0^2 - a$, so $\sqrt{a} - 2^{-1} \leqslant \sqrt{a} \leqslant b_0$. For the bound above, note that $(b_0 - 2^{-1})^2 = \frac{a^2}{4} \leqslant a$, so $b_0 \leqslant \sqrt{a} + 2^{-1}$. Finally, note that $1 \leqslant a$, so $1 \leqslant b_0$.

Suppose $n = m + 1$. Let $\delta \stackrel{\text{def}}{=} 2^{-2^m}$ and $c \stackrel{\text{def}}{=} \frac{a+b_m^2}{2b_m}$. Assume by induction that $\sqrt{a} - \delta \leqslant b_m \leqslant \sqrt{a} + \delta$ and $1 \leqslant b_m$. This means that $|b_m - \sqrt{a}| \leqslant \delta$. Therefore, $b_m^2 - 2b_m\sqrt{a} + a \leqslant \delta^2$. This implies that $c - \sqrt{a} \leqslant \frac{\delta^2}{2b_m} \leqslant \frac{\delta^2}{2}$ because $1 \leqslant b_m$. Also note that

$$0 \leqslant \frac{(b_m + \sqrt{a})^2}{2b_m} = c - \sqrt{a}.$$

Putting these together, one sees that $\sqrt{a} \leqslant c \leqslant \sqrt{a} + \frac{\delta^2}{2}$. Because b_{m+1} is within $\frac{\delta^2}{2}$ of c, we get $\sqrt{a} - \delta^2 \leqslant b_{m+1} \leqslant \sqrt{a} + \delta^2$. All that remains is to prove that $1 \leqslant b_{m+1}$. Now $1 \leqslant c$

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

because $1 \leqslant a$, and the only rational simpler than 1 is 0. Therefore, the only way b_{m+1} could be less than 1 is if it were 0. That would require that $1 \leqslant c \leqslant \frac{\delta^2}{2}$. But $\frac{\delta^2}{2}$ is less than 1, so c cannot be 0. □

For $0 < a$ that falls outside the interval $[1, 4[$, one can find some $m : \mathbb{Z}$ such that $1 \leqslant 4^m a < 4$. Therefore, one can define the square root function on the rationals as

$$\begin{array}{lll} & \sqrt{0} \stackrel{\text{def}}{=} 0 & \\ \text{For } 1 \leqslant a < 4, & \sqrt{a} \stackrel{\text{def}}{=} x & \text{(where } x \text{ is defined as in Theorem 31)} \\ \text{Otherwise,} & \sqrt{a} \stackrel{\text{def}}{=} \dfrac{\sqrt{4^m a}}{2^m} & \text{(for some } m : \mathbb{Z} \text{ such that } 1 \leqslant 4^m a < 4). \end{array}$$

Theorem 32. The function $\lambda a.\sqrt{a}$ is uniformly continuous from $\mathbb{Q}^{0+}$ to $\mathbb{R}$, with modulus $\lambda\varepsilon.\varepsilon^2$.

Using bind, the square root function can be lifted to the domain $\mathfrak{C}(\mathbb{Q}^{0+})$.

6.3. *More efficient power series*

A power series converges faster for values closer to 0. For sin and exp, the two equations mentioned earlier,

$$\begin{aligned} \exp(a) &\asymp \exp^2\left(\frac{a}{2}\right) \\ \sin(a) &\asymp 3\sin\left(\frac{a}{3}\right) - 4\sin^3\left(\frac{a}{3}\right), \end{aligned}$$

can be repeatedly applied to shrink the input arbitrarily close to 0. There is, of course, a trade-off between evaluating a polynomial each time this equation is used and having the power series converge faster. The optimal trade-off depends on the input and the implementation of the library. For simplicity, in my implementation I shrink the input until it is less than 2^{-50}, which is a value that seems to work well in my implementation for some example problems (Niqui and Wiedijk 2005).

Unfortunately, there do not seem to be any such nice reductions for ln or arctan.

6.4. *More efficient periodic functions*

The functions sin and cos are both periodic with period 2π. With a reasonably fast implementation of π, it is possible to subtract out a multiple of 2π to reduce the size of the argument to sin or cos.

$$\begin{aligned} \forall n : \mathbb{Z}, \sin(x) &\asymp \sin(x - n2\pi) \\ \forall n : \mathbb{Z}, \cos(x) &\asymp \cos(x - n2\pi) \end{aligned}$$

The best value of n would be $\left\lfloor \frac{x}{2\pi} \right\rceil$. The function mapping x to the nearest integer $\lfloor x \rceil$ is not computable for real numbers, but is computable for the rationals – it does not matter which integer $\frac{1}{2}$ maps to. The approximation $\frac{x}{2\pi}(\frac{1}{2})$ is within $\frac{1}{2}$ of $\frac{x}{2\pi}$, so $\left\lfloor \frac{x}{2\pi}(\frac{1}{2}) \right\rceil$ is an integer within 1 of $\left\lfloor \frac{x}{2\pi} \right\rceil$, which is good enough to use for n.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

In practice, one can be even cleverer with the symmetries of sin and cos to reduce the range a little bit further.

6.5. *Summing lists*

It is not uncommon to want to sum a list of real numbers. The use of ap in the definition of addition means that to compute $x + y$ within ε, one must compute both x and y within $\frac{\varepsilon}{2}$. If one associates the sum of a list of numbers all to one side $a_0 + (a_1 + (a_2 + \ldots + a_n))$, the first term a_0 is only approximated within $\frac{\varepsilon}{2}$, while the last term a_n is approximated within $\frac{\varepsilon}{2^n}$.

If one is summing a list with exactly a power of 2 number of terms, as in the definition of π, then one can associate the sum into a balanced binary tree. In this case all terms will be approximated to the same degree, but for handling more general cases, it is useful to define the sum of n real numbers directly:

$$\sum_{i=0}^{n} x_i \stackrel{\text{def}}{=} \lambda\varepsilon. \sum_{i=0}^{n} x_i\left(\frac{\varepsilon}{n}\right).$$

Theorem 33. The term $\sum_{i=0}^{n} x_i$ is a regular function.

7. Implementation

The main goal of this work is to produce an actual implementation of the real numbers that is reasonably efficient and, more importantly, reliable. I have written a prototype Haskell module that implements the functions described in this paper. This code, entitled *Few Digits* (O'Connor 2005), competed in the *'Many Digits' Friendly Competition* (Niqui and Wiedijk 2005) held on 3–4 October 2005. The library consists of fewer than 440 lines of Haskell code and is included as an appendix to this paper. Although the code did not do particularly well in the contest, finishing eighth out of nine, it can still compute thousands of decimal digits within minutes or seconds for many of the problems.

A lot of time is spent computing approx(a, ε). One suggestion I have received is that instead of spending so much time finding the perfect approximation, just find some good approximation quickly. Another place where much time is spent is in evaluating the polynomial approximation of the elementary functions. It only takes a polynomial of a few degrees to cause the exact evaluation on large (in terms of simplicity) rational numbers to become very slow. Such precise computation over the rationals is unnecessary because the resulting rational number will just be fed into compress, and a simpler rational number found. One possible solution may be to define the reals to be the completion of some dense set other than $\mathbb{Q}$ (such as the dyadic rationals $\mathbb{D} \stackrel{\text{def}}{=} \{\frac{a}{2^n} | an : \mathbb{Z}\}$). However, such a set is not closed under division, so evaluating Taylor series becomes more difficult.

The code is more or less faithful to the theory described in this paper. In the Haskell code, `Complete` is the type constructor of the monad, unit is called `const`, join is called `completeComplete`, map is called `evalUniformCts`, and bind is called `evalUniformCts2`. Unfortunately, because only uniformly continuous functions can be mapped, this monad does not fit into Haskell's `Monad` class.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Table 1. *Timings for the computation of two Haskell real number implementations.*

Implementation	$\cos(\frac{F_4}{F_5})$	$\cos(\frac{F_{2394}}{F_{2395}})$
Few Digits	0.93 s	37.53 s
Era	2.73 s	2.71 s

The most significant difference between the description here and the implementation is that the power series for exp is computed for small positive inputs. Even though the series is not alternating, the series is still nice enough to get a good handle on its modulus of convergence. Another minor difference is that map2, called `evalBinaryUniformCts`, is implemented directly rather than implemented via ap.

There are a couple of other Haskell specific issues to note. In order to make the reals an instance of the `Floating` class, it is necessary to make a new data type. This new data type is called `CReal`, which means *constructive reals* (or *computable reals* or *Cauchy reals*). Since a new data type must be created anyway, I also added a memoised integer approximation to the data structure.

7.1. *Comparison with other implementations*

Boehm, Cartwright, Riggle and O'Donnell were the first to implement an exact real arithmetic library (Boehm *et al.* 1986). They implemented both a functional representation and a representation as lazy lists. The real number representation in this paper is a functional representation. Lester has a very short Haskell module (a little over 200 lines) called Era (exact real arithmetic) that has a real number data type very similar to Boehm *et al.*'s implementation. Computer proofs of some of the algorithms have been verified by PVS (Lester and Gowland 2002; Lester 2003). Because of the similarity between the two programs, a quick comparison is in order (unfortunately, Era did not compete in the *'Many Digits' Friendly Competition*). Here is one test that illustrates the difference between the two implementations. Consider computing an approximation of $\cos(\frac{F_4}{F_5})$ and $\cos(\frac{F_{2394}}{F_{2395}})$ within 10^{-1000}, where F_n is the n^{th} Fibonacci number ($\frac{F_4}{F_5} = \frac{3}{5}$). The two ratios are both close to the golden ratio, so they are close to each other. Table 1 shows the time each system takes to do the real number portion of the computation (the Fibonacci ratios were precomputed).

Because Era uses dyadic rationals in its representation, the two ratios are equally bad for it. So Era takes about the same amount of time for both problems. The power series implementation in Few Digits does polynomial computation with rational numbers before compressing. The more terms and the larger the rational numbers, the longer the computation takes.

Another comparable implementation is Muñoz and Lester's PVS implementation (Muñoz and Lester 2005), which uses interval arithmetic to perform real number computation. Dealing with non-monotonic functions is difficult with interval arithmetic, and because of this, their implementation cannot approximate some expressions, such as $\sin(\frac{\pi}{2})$, to arbitrary accuracy. The approach in this paper has no problems dealing with non-monotonic functions.

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

8. Future work

In order to produce a highly reliable implementation of the real numbers, the next step is to write the proofs in a computer proof assistant such as Coq (The Coq development team 2004) and verify the correctness of the algorithms. Coq is a proof assistant based on the calculus of inductive constructions. It can be seen as a dependently typed functional programming language with inductive and coinductive data types. I believe that the theorems presented in this paper are sufficiently straightforward to be easily proved in a computer proof assistant.

9. Conclusion

The completion monad offers a simple, clear and convenient way of constructively creating the completion of any metric space, and for defining continuous functions on prelength spaces. I have created one possible implementation of the real numbers using this style, and have demonstrated that this approach can perform practical computations. This approach avoids some difficulties that other computer verified implementations have. This completion monad is flexible enough to allow for alternative implementations of real numbers, such as the completion of the dyadic rationals, and alternative implementations of elementary functions. This same monad could be used to create other complete spaces, such as the complex numbers, the unit sphere and the projective plane.

Appendix A. Haskell Source Code for *Few Digits*

```
module Complete where

type Gauge = Rational -- Intended to be strictly positive

-- Intended that d (f x) (f y) <= x + y
type Complete a = Gauge -> a

completeComplete :: (Complete (Complete a)) -> Complete a
completeComplete f eps = (f (eps/2)) (eps/2)

-- A uniformly continuous function on some subset of a to b
-- Hopefully the name of the function gives an indication of
-- the domain.
data UniformCts a b = UniformCts
        {modulus :: (Gauge -> Gauge),
         forgetUniformCts :: (a -> b)}

evalUniformCts :: UniformCts a b -> Complete a -> Complete b
evalUniformCts (UniformCts mu f) x eps = f (x (mu eps))

evalUniformCts2 :: UniformCts a (Complete b) ->
                   Complete a -> Complete b
evalUniformCts2 f x = completeComplete $ evalUniformCts f x

completeUniformCtsRange :: Complete (UniformCts a b) ->
                           UniformCts a (Complete b)
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
completeUniformCtsRange f = UniformCts mu g
 where
  mu eps = modulus (f (eps/2)) (eps/2)
  g x eps = forgetUniformCts (f eps) x

evalBinaryUniformCts :: UniformCts a (UniformCts b c) ->
                        Complete a -> Complete b -> Complete c
{- This was the original implementation.
evalBinaryUniformCts f x =
  evalUniformCts2 $
   completeUniformCtsRange $ evalUniformCts f x
-}

{- This implementation seems better -}
evalBinaryUniformCts f x y eps =
  (evalUniformCts approxf y) (eps/2)
 where
  approxf = (evalUniformCts f x) (eps/2)

module Base where

import Ratio

type Base = Rational

approxBase :: Base -> Rational -> Base
approxBase = approxRational

powers x = zipWith (%) (genpowers n) (genpowers d)
  where
   n = numerator x
   d = denominator x
   genpowers x = 1:(map (x*) (genpowers x))

module CReal where

import Base
import Complete
import Control.Exception

radius :: Base
radius = 2^^(-51)

data CReal = CReal {approx :: Complete Base,
                    intApprox :: Integer}

bound x = fromInteger (1+(abs (intApprox x)))

makeCReal :: Complete Base -> CReal
makeCReal x = CReal x' n
 where
  n = round (x (1/2))
  x' eps | eps >= 1 = fromInteger n
         | otherwise = x eps
{- produces a regular function whose resulting approximations
   are small in memory size -}
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
compress :: Complete Base -> Complete Base
compress x eps = approxBase (x (eps/2)) (eps/2)

squish :: CReal -> Complete Base
squish = compress . approx

instance Show CReal where
 show x = error "Cannot show a CReal"
 {- show x = show $ map (\n -> squish x ((1/2)^n)) [0..] -}

realBase :: Base -> CReal
realBase x = makeCReal (const x)

approxRange :: CReal -> Gauge -> (Base, Base)
approxRange x eps = (r-eps, r+eps)
 where
  r = approx x eps

{- proveNonZeroFrom will not terminate if the input is 0 -}
{- Finds some y st 0 < (abs y) <= (abs x) -}
proveNonZeroFrom :: Gauge -> CReal -> Base
proveNonZeroFrom g r | high < 0 = high
                     | 0 < low = low
                     | otherwise = proveNonZeroFrom (g/2) r
 where
  (low, high) = approxRange r g

proveNonZero = proveNonZeroFrom 1

makeCRealFun :: (UniformCts Base Base) -> CReal -> CReal
makeCRealFun f x = makeCReal $ evalUniformCts f (squish x)

makeCRealFun2 :: (UniformCts Base (Complete Base)) ->
                 CReal -> CReal
makeCRealFun2 f x = makeCReal $ evalUniformCts2 f (squish x)

makeCRealBinFun :: (UniformCts Base (UniformCts Base Base)) ->
                   CReal -> CReal -> CReal
makeCRealBinFun f x y =
  makeCReal $ evalBinaryUniformCts f (squish x) (squish y)

negateCts = UniformCts id negate

realNegate :: CReal -> CReal
realNegate = makeCRealFun negateCts

plusBaseCts a = UniformCts id (a+)
realTranslate a = makeCRealFun (plusBaseCts a)

plusCts :: UniformCts Base (UniformCts Base Base)
plusCts = UniformCts id plusBaseCts

realPlus :: CReal -> CReal -> CReal
realPlus = makeCRealBinFun plusCts
instance Eq CReal where
 a==b = 0==proveNonZero (realPlus a (realNegate b))
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
multBaseCts 0 = UniformCts (const 1) (const 0)
multBaseCts a = UniformCts mu (a*)
 where
  mu eps = eps/(abs a)

realScale :: Base -> CReal -> CReal
realScale 0 = \_ -> realBase 0
realScale a = makeCRealFun (multBaseCts a)

{- \x -> (\y -> (x*y)) is uniformly continuous on the domain
   (abs y) <= maxy -}
multUniformCts :: Base ->
                  UniformCts Base (UniformCts Base Base)
multUniformCts maxy = UniformCts mu multBaseCts
 where
  mu eps = assert (maxy>0) (eps/maxy)

{- We need to bound the value of x or y.  I think it is better
   to bound x so I actually compute y*x -}
realMult :: CReal -> CReal -> CReal
realMult x y = makeCRealBinFun (multUniformCts (bound x)) y x

absCts = UniformCts id abs

realAbs :: CReal -> CReal
realAbs = makeCRealFun absCts

instance Num CReal where
 (+) = realPlus
 (*) = realMult
 negate = realNegate
 abs = realAbs
 signum x = realScale (signum (proveNonZero x)) (realBase 1)
 fromInteger = realBase . fromInteger

{- domain is (-inf, nonZero] if nonZero < 0
   domain is [nonZero, inf) if nonZero > 0 -}
recipUniformCts :: Base -> UniformCts Base Base
recipUniformCts nonZero = UniformCts mu f
 where
  f a | 0 <= nonZero = recip (max nonZero a)
      | otherwise = recip (min a nonZero)
  mu eps = eps*(nonZero^2)

realRecipWitness :: CReal -> Base -> CReal
realRecipWitness x nonZero =
  makeCRealFun (recipUniformCts nonZero) x

realRecip :: CReal -> CReal
realRecip x = realRecipWitness x (proveNonZero x)

instance Fractional CReal where
 recip = realRecip
 fromRational = realBase . fromRational
intPowerCts _ 0 = UniformCts (const 1) (const 1)
intPowerCts maxx n = UniformCts mu (^n)
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
 where
  mu eps = assert (maxx > 0) $ eps/((fromIntegral n)*(maxx^(n-1)))

realPowerInt x n = makeCRealFun (intPowerCts (bound x) n) x

type Polynomial a = [a]

evalPolynomial :: (Num a) => Polynomial a -> a -> a
evalPolynomial [] x = 0
evalPolynomial (a:as) x = a + x*(evalPolynomial as x)

diffPolynomial :: (Num a) => Polynomial a -> Polynomial a
diffPolynomial p = zipWith (*) (tail p) (map fromInteger [1..])

polynomialUniformCts :: Base ->
                        Polynomial Base -> UniformCts Base Base
polynomialUniformCts _ [] = UniformCts (const 1) (const 0)
polynomialUniformCts maxx p |maxSlope==0 =
                               UniformCts (const 1) (const (head p))
                            |otherwise =
                               UniformCts mu (evalPolynomial p)
 where
  maxSlope = evalPolynomial (map abs (diffPolynomial p)) (max 1 maxx)
  mu eps = assert (maxSlope > 0) $ eps/maxSlope

realBasePolynomial :: Polynomial Base -> CReal -> CReal
realBasePolynomial p x =
 makeCRealFun (polynomialUniformCts (bound x) p) x

factorials = fact 1 1
 where
  fact i j = i:(fact (i*j) (j+1))

interleave [] _ = []
interleave (x:xs) l = x:(interleave l xs)

taylorApprox m p x eps =
  sum (zipWith (*) p (takeWhile highError preTerms))
 where
  preTerms = zipWith (/) (powers x) factorials
  highError t = m*(abs t) >= eps

{- only valid for x <= ln(2).  Works best for |x| <= 1/2 -}
rationalSmallExp :: Base -> CReal
rationalSmallExp x = assert ((abs x)<=(1/2)) $
  makeCReal $ expTaylorApprox
 where
  m | x <= 0 = 1
    | otherwise = 2
  expTaylorApprox eps =
    sum terms
   where
    terms = takeWhile highError $ zipWith (/) (powers x) factorials
    highError t = m*(abs t) >= eps
rationalExp :: Base -> Base -> CReal
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
rationalExp tol x | (abs x) <= tol = rationalSmallExp x
                  | otherwise = realPowerInt (rationalExp tol (x/2)) 2

expUniformCts :: Integer -> UniformCts Base (Complete Base)
expUniformCts upperBound = UniformCts mu (approx . rationalExp radius)
 where
  mu eps | upperBound <= 0 = eps*(2^(-upperBound))
         | otherwise = eps/(3^upperBound)

realExp :: CReal -> CReal
realExp x = makeCRealFun2 (expUniformCts (1+intApprox x)) x

{- Requires that abs(a!!i+1) < abs(a!!i) and the sign of the
   terms alternate -}
alternatingSeries :: [Base] -> Complete Base
alternatingSeries a eps = sum partSeries
 where
  partSeries = (takeWhile (\x -> (abs x) > eps) a)

rationalSin :: Base -> Base -> CReal
rationalSin tol x | tol <= (abs x) =
                     realBasePolynomial [0, 3, 0, (-4)]
                     (rationalSin tol (x/3))
                  | otherwise = CReal (alternatingSeries series) 0
 where
  series = fst $ unzip $ iterate
            (\(t,n) -> (-t*(x^2)/(n^2+n),n+2)) (x, 2)

sinCts :: UniformCts Base (Complete Base)
sinCts = UniformCts id (approx . rationalSin radius)

realSlowSin :: CReal -> CReal
realSlowSin = makeCRealFun2 sinCts

realSin :: CReal -> CReal
realSin x | 0==m = realSlowSin x'
          | 1==m = realSlowCos x'
          | 2==m = negate $ realSlowSin x'
          | 3==m = negate $ realSlowCos x'
 where
  n = intApprox (x / realPi2)
  m = n `mod` 4
  x' = x - (realScale (fromInteger n) realPi2)

rationalCos :: Base -> Base -> CReal
rationalCos tol x = realBasePolynomial [1, 0, (-2)]
  (rationalSin tol (x/2))

cosCts :: UniformCts Base (Complete Base)
cosCts = UniformCts id (approx . rationalCos radius)

realSlowCos :: CReal -> CReal
realSlowCos = makeCRealFun2 cosCts

realCos :: CReal -> CReal
realCos x | 3==m = realSlowSin x'
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
          | 0==m = realSlowCos x'
          | 1==m = negate $ realSlowSin x'
          | 2==m = negate $ realSlowCos x'
 where
  n = intApprox (x / realPi2)
  m = n `mod` 4
  x' = x - (realScale (fromInteger n) realPi2)

{- computes ln(x).  only valid for 1<=x<2 -}
rationalSmallLn :: Base -> CReal
rationalSmallLn x = assert (1<=x && x<=(3/2)) $
  makeCReal $
  alternatingSeries (zipWith (*) (poly 1) (tail (powers (x-1))))
 where
  poly n = (1/n):(-1/(n+1)):(poly (n+2))

{- requires that 0<=x -}
rationalLn :: Base -> CReal
rationalLn x | x<1 = negate (posLn (recip x))
             | otherwise = posLn x
 where
  ln43 = rationalSmallLn (4/3)
  ln2 = wideLn 2
  {- good for 1<=x<=2 -}
  wideLn x | x < (3/2) = rationalSmallLn x
           | otherwise = (rationalSmallLn ((3/4)*x)) + ln43
  {- requires that 1<=x -}
  posLn x | n==0 = wideLn x
          | otherwise = (wideLn x') + (realScale n ln2)
   where
    (x',n) = until (\(x,n) -> (x<=2)) (\(x,n) -> (x/2,n+1)) (x,0)

{- domain is [nonZero, inf) -}
lnUniformCts :: Base -> UniformCts Base (Complete Base)
lnUniformCts nonZero = UniformCts mu f
 where
  f x = approx $ rationalLn (max x nonZero)
  mu eps = assert (nonZero > 0) $ eps*nonZero

realLnWitness :: CReal -> Base -> CReal
realLnWitness x nonZero = makeCRealFun2 (lnUniformCts nonZero) x

realLn :: CReal -> CReal
realLn x = realLnWitness x (proveNonZero x)

{- only valid for (abs x) < 1 -}
rationalSmallArcTan :: Base -> CReal
rationalSmallArcTan x = assert ((abs x)<(1/2)) $ makeCReal $
  alternatingSeries (zipWith (\x y->x*(y^2)) (series 0) (powers x))
 where
  series n = (x/(n+1)):(-x/(n+3)):(series (n+4))

rationalArcTan :: Base -> CReal
rationalArcTan x | x <= (-1/2) = negate $ posArcTan $ negate x
                 | otherwise = posArcTan x
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
 where
  {-requires (-1/2) < x-}
  posArcTan x | 2 < x = realPi2 - rationalSmallArcTan (recip x)
              | (1/2) <= x = realPi4 + rationalSmallArcTan y
              | otherwise = rationalSmallArcTan x
   where
    y = (x-1)/(x+1)

arcTanCts :: UniformCts Base (Complete Base)
arcTanCts = UniformCts id (approx . rationalArcTan)

realArcTan :: CReal -> CReal
realArcTan = makeCRealFun2 arcTanCts

{- Computes x * Pi -}
{- http://aemes.mae.ufl.edu/~uhk/PI.html -}
scalePi :: Base -> CReal
scalePi x =
 ((realScale (x*48) (rationalSmallArcTan (1/38))) +
  (realScale (x*80) (rationalSmallArcTan (1/57)))) +
 ((realScale (x*28) (rationalSmallArcTan (1/239))) +
  (realScale (x*96) (rationalSmallArcTan (1/268))))

real2Pi = scalePi 2
realPi = scalePi 1
realPi2 = scalePi (1/2)
realPi4 = scalePi (1/4)

nestedBalls :: [(Base, Base)] -> Complete Base
nestedBalls ((lb,ub):bs) eps | (ub - lb) < 2*eps = (ub+lb)/2
                             | otherwise = nestedBalls bs eps

{- My algorithm -}
{-
rationalSqrt :: Base -> CReal
rationalSqrt x = makeCReal
  (nestedBalls (iterate betterBounds (0,(1+x)/2)))
 where
  betterBounds (lb, ub) =
    assert (lb <= lb' && lb' <= ub' && ub' <= ub) $ (lb', ub')
   where
    intercept a b = (x + a*b)/(a+b)
    lb' = intercept lb ub
    ub1 = intercept lb lb
    ub2 = intercept ub ub
    ub' = if (lb > 0) then (min ub1 ub2) else ub2
-}
{- Freek's algorithm -}
rationalSqrt :: Base -> CReal
rationalSqrt n | n < 1 = realScale (1/2) (rationalSqrt (4*n))
               | 4 <= n = realScale 2 (rationalSqrt (n/4))
               | otherwise = makeCReal (\eps -> f eps)
 where
  f eps = fst $ until (\(x,err) -> err <= eps) newton ((1+n)/2, 1/2)
  newton (x,err) = (approxBase x' e1, e1*2)
```

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

```
   where
    x' = (n+x^2)/(2*x)
    e1 = err^2/2

sqrtCts :: UniformCts Base (Complete Base)
sqrtCts = UniformCts (^2) (approx . rationalSqrt)

realSqrt :: CReal -> CReal
realSqrt = makeCRealFun2 sqrtCts

instance Floating CReal where
 exp = realExp
 log = realLn
 pi = realPi
 sin = realSin
 cos = realCos
 atan = realArcTan
 sqrt = realSqrt
 sinh x = realScale (1/2) (exp x - (exp (-x)))
 cosh x = realScale (1/2) (exp x + (exp (-x)))
 asin x = atan (x/sqrt(realTranslate 1 (negate (realPowerInt x 2))))
 acos x = realPi2 - asin x
 acosh x = log (x+sqrt(realTranslate (-1) (realPowerInt x 2)))
 asinh x = log (x+sqrt(realTranslate 1 (realPowerInt x 2)))
 atanh x = realScale (1/2)
   (log ((realTranslate 1 x) / (realTranslate 1 (negate x))))

{- testing stuff is below -}
test0 = makeCReal id

answer n x = shows (intApprox (realScale (10^n) x))
  "x10^-"++(show n)

sumRealList :: [CReal] -> CReal
sumRealList [] = realBase 0
sumRealList l = makeCReal
  (\eps -> sum (map (\x -> approx x (eps/n)) l))
 where
  n = fromIntegral $ length l
```

Acknowledgements

I would like to thank Milad Niqui for convincing me to write a Haskell implementation for his contest. I would like to thank Freek Wiedijk for advising me on using Newton's method to compute square roots. Most importantly, I would like to thank my advisor Bas Spitters for his advice while I was developing these ideas.

This document was originally produced using TEXMACS.

References

Bishop, E. and Bridges, D. (1985) Constructive Analysis. *Grundlehren der mathematischen Wissenschaften* **279**, Springer-Verlag

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press

Boehm, H.-J., Cartwright, R., Riggle, M. and O'Donnell, M. J. (1986) Exact real arithmetic: a case study in higher order programming. In: *LFP '86: Proceedings of the 1986 ACM conference on LISP and functional programming*, ACM Press 162–173.

Burago, D., Burago, Y. and Ivanov, S. (2001) *A Course in Metric Geometry*, Graduate Studies in Mathematics **33**, American Mathematical Society.

Cruz-Filipe, L. (2003) A constructive formalization of the fundamental theorem of calculus. In: Geuvers, H. and Wiedijk, F. (eds.) Types for Proofs and Programs. *Springer-Verlag Lecture Notes in Computer Science* **2646** 108–126.

Cruz-Filipe, L. and Spitters, B. (2003) Program extraction from large proof developments. In: Basin, D. and Wolff, B. (eds.) Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003. *Springer-Verlag Lecture Notes in Computer Science* **2758** 205–220.

Hales, T. C. (2002) A computer verification of the Kepler conjecture. In: *Proceedings of the International Congress of Mathematicians, Beijing, 2002*, Volume III, Higher Ed. Press 795–804.

Kurzweg, U. H. (2001) Pi. (Available at `http://aemes.mae.ufl.edu/~uhk/PI.html`.)

Lester, D. (2003) Using PVS to validate the inverse trigonometric functions of an exact arithmetic. In: Alt, R., Frommer, A., Kearfott, R. B. and Luther, W. (eds.) Numerical Software with Result Verification. *Springer-Verlag Lecture Notes in Computer Science* **2991** 259–273.

Lester, D. and Gowland, P. (2002) Using PVS to validate the algorithms of an exact arithmetic. *Theoretical Computer Science* **291** (2) 203–218.

Moggi, E. (1991) Notions of computation and monads. *Information and Computation* **93** (1) 55–92.

Muñoz, C. and Lester, D. (2005) Real number calculations and theorem proving. In: Hurd, J. and Melham, T. (eds.) Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2005. *Springer-Verlag Lecture Notes in Computer Science* **3603** 195–210.

Niqui, M. and Wiedijk, F. (2005) The 'many digits' friendly competition 2005. (Details at `http://www.cs.ru.nl/\~milad/manydigits`.)

O'Connor, R. (2005) Few digits 0.4.0. (Available at `http://r6.ca/FewDigits/`.)

Odlyzko, A. M. and te Riele, H. J. J. (1985) Disproof of the Mertens conjecture. *J. Reine Angew. Math.* **357** 138–160.

Peyton Jones, S. (ed.) (2003) *Haskell 98 Language and Libraries: The Revised Report*, Cambridge University Press.

The Coq development team (2004) *The Coq proof assistant reference manual*, Version 8.0, LogiCal Project.

Wadler, P. (1992) Monads for functional programming. In: *Proceedings of the Marktoberdorf Summer School on Program Design Calculi.*

https://doi.org/10.1017/S0960129506005871 Published online by Cambridge University Press