Abstract Datatypes for Real Numbers in Type Theory

Citation for published version:
Escardó, MH & Simpson, A 2014, Abstract Datatypes for Real Numbers in Type Theory. in Rewriting and Typed Lambda Calculi: Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. vol. 8560, Springer International Publishing, pp. 208-223. DOI: 10.1007/978-3-319-08918-8_15

Digital Object Identifier (DOI):
10.1007/978-3-319-08918-8_15

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Rewriting and Typed Lambda Calculi

Abstract Datatypes for Real Numbers in Type Theory

Martín Hötzel Escardó[1] and Alex Simpson[2]

[1] School of Computer Science, University of Birmingham
[2] LFCS, School of Informatics, University of Edinburgh

Abstract. We propose an abstract datatype for a closed interval of real numbers to type theory, providing a representation-independent approach to programming with real numbers. The abstract datatype requires only function types and a natural numbers type for its formulation, and so can be added to any type theory that extends Gödel's System T. Our main result establishes that programming with the abstract datatype is equivalent in power to programming intensionally with representations of real numbers. We also consider representing arbitrary real numbers using a mantissa-exponent representation in which the mantissa is taken from the abstract interval.

1 Introduction

Exact real-number computation uses infinite representations of real numbers to compute exactly with them, avoiding round-off errors [16,2,3]. In practice, such representations can be implemented as streams or functions, allowing any computable (and hence *a fortiori* continuous) function to be programmed.

This approach of programming with representations of real numbers has drawbacks from the programmer's perspective. Great care must be taken to ensure that different representations of the same real number are treated equivalently. Furthermore, a programmer ought to be able to program with real numbers without knowing how they are represented, leading to more transparent programs, and also allowing the underlying implementation of real-number computation to be changed, e.g., to improve efficiency. In short, the programmer would like to program with an *abstract datatype of real numbers*.

Various interfaces for an abstract datatype for real numbers have been investigated in the context of typed functional programming languages based on PCF, e.g., [6,7,10,8,1,9], making essential use of the presence of general recursion. In this paper, we consider the more general scenario of typed functional programming with primitive recursion. This generality has the advantage that it can be seen as a common core both to standard functional programming languages with general recursion (ML, Haskell, etc.), and also to the type theories used in dependently-typed programming languages such as Agda [4], and proof assistants such as Coq [13], in which all functions are total.

To maximize generality, we keep the type theory in this paper as simple as possible. Our base calculus is just simply-typed λ-calculus with a base type of natural numbers, otherwise known as Gödel's System T. To this, we add a new type constant I, which acts as an abstract datatype for the interval $[-1,1]$ of real numbers, together with an associated interface of basic operations. Our main result (Theorem 1) establishes that programming with the abstract datatype is equivalent in power to programming intensionally with representations of reals.

The development in this paper builds closely on our LICS 2001 paper [11], where we gave a category-theoretic universal property for the interval $[-1,1]$. The interface we provide for the type I is based directly on the universal property defined there. In [11], the definability power of the universal property was already explored, to some extent, via a class of *primitive interval functions* on $[-1,1]$, named by analogy to the primitive recursive functions. The role of a crucial doubling function was identified, relative to which all continuous functions on $[-1,1]$ were shown to be primitive-interval definable relative to oracles $\mathbb{N} \to \mathbb{N}$.

The new departure of the present paper is to exploit these ideas in a type-theoretic context. The cumbersome definition of primitive interval functions is replaced by a very simple interface for the abstract datatype I (Sect. 3). The role of the doubling function is again crucial, with its independence from the other constants of the interface now being established by a logical relations argument (proof of Prop. 4). And the completeness of the interface once doubling is added (Theorem 1) is now established relative to the setting at hand (System T computability) rather than relative to oracles (Sect. 4). In addition, we show that the type theoretic framework provides a natural context for proving equalities between functions on reals (based on the equalities in Fig. 2), and for programming on the full real line $\mathbb{R}$ via a mantissa-exponent representation (Sect. 5).

2 Real-number Computation in System T

In this section, we recall how exact real-number computation is rendered possible by choosing an appropriate representation of real numbers. A natural first attempt would be to represent real numbers using streams or functions to implement one of the standard digit representations (decimal, binary, etc.). For example, a real number in $[0,1]$ would be represented via a binary expansion as an infinite sequence of 0s and 1s. As is well known (see, e.g., [5,6,7]), however, such representations makes it impossible to compute even simple functions (on the interval) such as binary average on real numbers. The technical limitation here is that there is no continuous function $\{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$ that given sequences α, β as input, representing $x, y \in [0,1]$ respectively, returns a representation of $\frac{x+y}{2}$ as result. In general, it is impossible to return even a single output digit without examining all (infinitely many) input digits.

This problem is avoided by choosing a different representation. To be appropriate for computation, any representation must be *computably admissible* in the sense of [15]. Each of the examples below is a computably admissible representation of real numbers, in the interval $[-1,1]$, using streams:

$$q_0 : q_1 : q_2 : q_3 : q_4 : q_5 : \dots$$

```
type I = [Int]  -- Represents [-1,1] in binary using digits -1,0,1.
minusOne, one :: I
minusOne  = repeat (-1)
one       = repeat 1
type J = [Int]  -- Represents [-n,n] in binary using digits |d| <= n
divideBy :: Int -> J -> I
divideBy n (a:b:x) = let d = 2*a+b
                       in if d < -n then -1 : divideBy n (d+2*n:x)
                     else if d >  n then  1 : divideBy n (d-2*n:x)
                                    else  0 : divideBy n (d:x)
mid :: I -> I -> I
mid x y = divideBy 2 (zipWith (+) x y)
bigMid :: [I] -> I
bigMid = (divideBy 4).bigMid'
  where bigMid'((a:b:x):(c:y):zs) = 2*a+b+c : bigMid'((mid x y):zs)
affine :: I -> I -> I -> I
affine a b x = bigMid [h d | d <- x]
  where h (-1) = a
        h   0  = mid a b
        h   1  = b
```

Fig. 1. Haskell programs using signed binary notation

of discrete data.

1. *Fast Cauchy sequences:* Require q_i to be rational numbers in $[-1, 1]$ such that $|q_{i+1} - q_i| \leq 2^{-i}$ for all i. The stream represents the real number $\lim_i q_i$.
2. *Signed binary:* Require $q_i \in \{-1, 0, 1\}$. The stream represents the real number $\sum_{i \geq 0} 2^{-(i+1)} q_i$.

Many other variations are possible. Crucially, all computably admissible representations are computably interconvertible. For representations used in practice, the conversions can be defined in System T.

Both to illustrate the style of programming that is required with such representations, and for later reference, Fig. 1 presents some simple functions on real numbers, in Haskell, using signed binary notation. The code defines a type `I` for the interval $[-1, 1]$, and constants `one` and `minusOne` of type `I`, for the streams `1:1:1:1:1:...` and `-1:-1:-1:-1:-1:...`, which represent 1 and -1 respectively. The function `mid` represents the binary average function, for which we use a convenient algebraic notation:

$$x \oplus y \;=\; \frac{x+y}{2} \;\; .$$

The function `bigMid` maps infinite streams of real numbers to real numbers, and represents the function $\mathrm{M}\colon [-1, 1]^\omega \to [-1, 1]$ defined by

$$\mathrm{M}((x_n)_n) \;=\; \sum_{n \geq 0} \frac{x_n}{2^{n+1}} \;\; .$$

Finally, `affine` represents $\mathrm{aff}\colon [-1,1] \to [-1,1] \to [-1,1] \to [-1,1]$ defined by

$$\mathrm{aff}\, x\, y\, z \;=\; \frac{(1-z)\,x + (1+z)\,y}{2} \enspace .$$

The type `J` and function `divideBy` just provide auxiliary machinery.

At this point, the selection of example functions in Fig. 1 will appear peculiar. The reasons behind the choice will be clarified in Sect. 3.

Although presented in Haskell, the above algorithms can be formalized in almost any type theory containing a type of natural numbers and function types. Moreover, the recursive structure of the algorithms is tame enough to be formulated using primitive recursion. Thus a natural basic type theory for studying this approach to real number computation is Gödel's System T, see, e.g., [12], which is simply-typed λ-calculus with a natural numbers type with associated primitive recursion operator. Since this type theory will form the basis of the rest of the paper we now review it in some detail.

Types (we include product types for convenience in Sect. 5) are given by:

$$\sigma \;::=\; \mathsf{N} \mid \sigma \times \tau \mid \sigma \to \tau \enspace .$$

These have a set-theoretic semantics with types being interpreted by their set-theoretic counterparts:

$$[\![\mathsf{N}]\!] = \mathbb{N} \qquad [\![\sigma \times \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!] \qquad [\![\sigma \to \tau]\!] = [\![\sigma]\!] \to [\![\tau]\!] \enspace .$$

We use standard notation for terms of the simply-typed λ-calculus; e.g., we write $\Gamma \vdash t\colon \tau$ to mean that term t has type τ in type context Γ. The constants associated with the type N are:

$$0 \;:\; \mathsf{N} \qquad \mathsf{s} \;:\; \mathsf{N} \to \mathsf{N} \qquad \mathsf{primrec}_\sigma \;:\; \sigma \to (\sigma \to \mathsf{N} \to \sigma) \to \mathsf{N} \to \sigma$$

with semantics defined by:

$$\begin{aligned} [\![0]\!] &= 0 & [\![s]\!]\; n &= n+1 \\ [\![\mathsf{primrec}]\!]\; x\; f\; 0 &= x & [\![\mathsf{primrec}]\!]\; x\; f\; (n+1) &= f\; ([\![\mathsf{primrec}]\!]\; x\; f\; n)\; n \enspace . \end{aligned}$$

We have two main interests in System T. The first is that it serves as a basic functional programing language, for which, the standard strongly normalizing and confluent β-reduction relation is used. The second is that System T serves as the basis of a formal system for reasoning about equality between functions. For this, we introduce axioms and rules for deriving *typed equations* of the form $\Gamma \vdash t = u : \sigma$ between terms t, u such that $\Gamma \vdash t : \sigma$ and $\Gamma \vdash u : \sigma$. These rules include the usual ones asserting that equality is a typed congruence relation. Also, whenever $\Gamma \vdash t : \sigma$ and t β-reduces to u, we have an equation $\Gamma \vdash t = u : \sigma$. Finally, we add extensionality equations, which comprise the usual η-equalities for product and function types, together with the rule below, which asserts the uniqueness of the $\mathsf{primrec}$ iterator.

$$\frac{\Gamma \vdash t\; u\; v\; 0 = u : \sigma \qquad \Gamma, x{:}\mathsf{N} \vdash t\; u\; v\; (\mathsf{s}(x)) \;=\; v\; u\; (t\; u\; v\; x) : \sigma}{\Gamma \vdash t\; u\; v \;=\; \mathsf{primrec}\; u\; v : \mathsf{N} \to \sigma}$$

(The types of the component terms are not stated explicitly since they can be inferred from the context.)

The term language of System T, in the version we are considering, can be interpreted in any cartesian-closed category with natural numbers object; and our equational rules are sound and complete with respect to such interpretations.

Exact-real-number computation can be carried out in System T by encoding any of the usual computably admissible representation of $[-1,1]$ using the type $\mathsf{N} \to \mathsf{N}$, and these representations are all interconvertible in System T.

We examine just the case of signed binary in detail. We consider a function $\alpha\colon \mathbb{N} \to \mathbb{N}$ as encoding the signed binary stream

$$((\alpha(0) \bmod 3){-}1) : ((\alpha(1) \bmod 3){-}1) : ((\alpha(2) \bmod 3){-}1) : ((\alpha(3) \bmod 3){-}1) : \ldots$$

All the Haskell programs in Fig. 1 are then routinely translatable into System T terms of appropriate type. (Just a little effort is needed to translate the general recursion into uses of $\mathsf{primrec}$.)

Consider the function real: $(\mathbb{N} \to \mathbb{N}) \to [-1,1]$ defined by

$$\mathrm{real}(\alpha) \;=\; \sum_{i \geq 0} 2^{-(i+1)}\,((\alpha(i) \text{ mod } 3) - 1)\ .$$

We say that a function $f\colon [-1,1]^k \to [-1,1]$ is *T-representable* if there exists a closed term $t\colon (\mathsf{N} \to \mathsf{N})^k \to \mathsf{N} \to \mathsf{N}$ making the following diagram commute.

$$\begin{tikzcd}
(\mathbb{N} \to \mathbb{N})^k \arrow[r, "{[\![t]\!]}"] \arrow[d, "\mathrm{real}^k"'] & (\mathbb{N} \to \mathbb{N}) \arrow[d, "\mathrm{real}"] \\
{[-1,1]^k} \arrow[r, "f"'] & {[-1,1]}
\end{tikzcd}$$

Since the vertical maps are topological quotients relative to the product (Baire space) topology on $\mathbb{N} \to \mathbb{N}$, and every T-definable $[\![t]\!]$ is continuous, it follows that every T-representable function f is continuous.

The programs in Fig. 1, when recast as System T terms, provide examples of representations of functions on reals. But programming in this style has the disadvantages discussed in Sect. 1. Accordingly, we now turn to the alternative approach of defining an abstract datatype for real numbers.

3 System I

We add a new type constant I to our base type theory to act as an abstract datatype for the closed interval $[-1,1]$ of real numbers. In the case of System T, we call the resulting extension System I; it has types

$$\sigma \;::=\; \mathsf{N} \mid \mathsf{I} \mid \sigma \times \tau \mid \sigma \to \tau\ ,$$

and we extend the set-theoretic semantics with the clause

$$[\![\mathsf{I}]\!] \;=\; [-1,1] \;\;.$$

The interface for the type will roughly implement the idea that the closed interval is determined as the free convex set on 2 generators (-1 and 1) with respect to affine maps. However, since the notion of *convexity* requires a pre-existing interval for its formulation, we replace convexity with the existence of *iterated midpoints* and we replace affineness with preservation of (iterated) midpoints, following [11].

The term language is generated by adding the following new typed constants. We pair each with its denotational interpretation in order to specify its intended meaning. In doing so, we make use of the functions defined in Sect. 2.

$$\begin{array}{rcl@{\qquad\qquad}rcl}
1 &:& \mathsf{I} & [\![1]\!] &=& 1 \\
-1 &:& \mathsf{I} & [\![-1]\!] &=& -1 \\
\mathsf{m} &:& \mathsf{I}\times\mathsf{I}\to\mathsf{I} & [\![\mathsf{m}]\!](x,y) &=& x\oplus y \\
\mathsf{M} &:& (\mathsf{N}\to\mathsf{I})\to\mathsf{I} & [\![\mathsf{M}]\!] &=& \mathrm{M} \\
\mathsf{aff} &:& \mathsf{I}\to\mathsf{I}\to\mathsf{I}\to\mathsf{I} & [\![\mathsf{aff}]\!] &=& \mathrm{aff}
\end{array}$$

We have here adopted a convention that we shall continue to follow of using sans-serif for λ-calculus constants and defined terms and using the same symbol in roman the corresponding mathematical function.

Figure 2 presents equational axioms and rules extending those for System T. A simple consequence of the equational rules for midpoints and iterated midpoints is that

$$x,y\!:\!I \vdash \mathsf{m}(x,y) = \mathsf{M}(x,y,y,y,y,\dots) : \mathsf{I} \;\;,$$

where we write $(x,y,y,y,y,\dots)$ as a convenient shorthand for the System I term $\mathsf{primrec}\,(x)\,(\lambda z:\mathsf{I}.\,\lambda n:\mathsf{N}.\,y)$, i.e., the function that is x at 0, and y at every natural number > 0. (Henceforth, we shall adopt other similar notational shorthands, without discussion.) Thus the constant m is redundant, and could be removed from the system. We include it, however, since the equations are more perspicuous with m included as basic. In fact, m is used frequently in the sequel, and we adopt the more suggestive notation $t\oplus u$ in preference to $\mathsf{m}(t,u)$.

We now develop some simple programming in System I, to explore its power as a programming language for defining real numbers, and functions on them.

$$\begin{array}{rcl}
0 &:=& (-1)\oplus 1 \\
-x &:=& \mathsf{aff}\ 1\ (-1)\ x \\
xy &:=& \mathsf{aff}\ (-x)\ x\ y \\
\dfrac{1}{3} &:=& \mathsf{M}(1,-1,\,1,-1,\,1,-1,\,1,-1,\dots)
\end{array}$$

More generally, any rational number is definable using M applied to an eventually periodic sequence of 1s and (-1)s. Even more generally, any real number with a System-T-definable binary expansion is definable.

(m) Midpoint equations.

$$\Gamma \vdash \mathsf{m}(t,t) = t : \mathsf{I} \qquad\qquad \Gamma \vdash \mathsf{m}(t,u) = \mathsf{m}(u,t) : \mathsf{I}$$

$$\Gamma \vdash \mathsf{m}(\mathsf{m}(t,u), \mathsf{m}(v,w)) = \mathsf{m}(\mathsf{m}(t,v), \mathsf{m}(u,w)) : \mathsf{I}$$

(M) Iterated midpoint equations.

$$\Gamma \vdash \mathsf{M}(t) = \mathsf{m}(t(0),\ \mathsf{M}(\lambda i{:}\mathsf{N}.\, t(i+1))) : \mathsf{I} \qquad \frac{\Gamma, i{:}\mathsf{N} \vdash t(i) = \mathsf{m}(u(i), t(i+1)) : \mathsf{I}}{\Gamma \vdash t(0) = \mathsf{M}(u) : \mathsf{I}}$$

(a) Equations for aff.

$$\Gamma \vdash \mathsf{aff}\, t\, u\, (-1) = t : \mathsf{I} \qquad\qquad \Gamma \vdash \mathsf{aff}\, t\, u\, 1 = u : \mathsf{I}$$

$$\Gamma \vdash \mathsf{aff}\, t\, u\, (\mathsf{m}(v,w)) = \mathsf{m}(\mathsf{aff}\ t\, u\, v,\ \mathsf{aff}\, t\, u\, w) : \mathsf{I}$$

$$\frac{\Gamma, x{:}\mathsf{I}, y{:}\mathsf{I} \vdash f\,(\mathsf{m}(x,y)) = \mathsf{m}(f(x),\ f(y)) : \mathsf{I}}{\Gamma \vdash f = \mathsf{aff}\,(f(-1))\,(f(1)) : \mathsf{I} \to \mathsf{I}}$$

(C) Cancellation

$$\frac{\Gamma \vdash \mathsf{m}(t,v) = \mathsf{m}(u,v) : \mathsf{I}}{\Gamma \vdash t = u : \mathsf{I}}$$

(E) Joint I-epimorphicity of $\mathsf{m}(\,\cdot\,,1)$ and $\mathsf{m}(\,\cdot\,,-1)$.

$$\frac{\Gamma, x{:}I \vdash f(\mathsf{m}(x,1)) = g(\mathsf{m}(x,1)) : \mathsf{I} \quad \Gamma, x{:}I \vdash f(\mathsf{m}(x,-1)) = g(\mathsf{m}(x,-1)) : \mathsf{I}}{\Gamma \vdash f = g : \mathsf{I} \to \mathsf{I}}$$

Fig. 2. Equations for System I

Proposition 1. *The following equalities are derivable from the axioms and rules in Fig. 2 (without using (M), (C) and (E)).*

$$\begin{array}{rclcrcl}
--x &=& x & \qquad & (x\,y)\,z &=& x\,(y\,z) \\
x \oplus -x &=& 0 & & x\,(-y) &=& -(x\,y) \\
x\,0 &=& 0 & & x\,(y \oplus z) &=& (x\,y) \oplus (x\,z) \\
x\,y &=& y\,x & & & &
\end{array}$$

So far, we have seen that the type I supports the arithmetic of multiplication and average, together with its expected equational properties. We now look at possibilities for defining functions that arise in analysis. Suppose we have a function f defined by a power series

$$f(x) \;=\; \sum_{n \geq 0} a_n\, x^n$$

where $a_n \in [-1,1]$. Then

$$\frac{1}{2} f\left(\frac{x}{2}\right) \;=\; \underset{n}{\mathrm{M}}\; a_n x^n \qquad\qquad \text{(which abbreviates } \mathrm{M}(\lambda n.\, a_n x^n)\text{)}.$$

As a consequence, using the arithmetic defined above, the following are all definable in System I.

$$\begin{aligned}
\frac{1}{2-x} \;&:=\; \underset{n}{\mathsf{M}}\, x^n \\
\frac{1}{2}\exp\left(\frac{x}{2}\right) \;&:=\; \underset{n}{\mathsf{M}}\; \frac{x^n}{n!} \\
\frac{1}{2}\cos\left(\frac{x}{2}\right) \;&:=\; \underset{n}{\mathsf{M}}\; (1-\mathrm{parity}(n))\,(-1)^{\frac{n}{2}}\,\frac{x^n}{n!}
\end{aligned}$$

These (and other similar) examples cover many functions from analysis, in versions with very particular scalings. We shall return to the issue of scaling below.

All functions defined above are continuous and smooth on $[-1,1]$. System I is also powerful enough to define non-smooth functions. We present two examples, exhibiting different degrees of non-smoothness. Define:

$$\begin{aligned}
\mathsf{times}^*(x,y) \;&:=\; \mathrm{aff}\ (-1)\ x\ y \\
\mathsf{sq}^*(x) \;&:=\; \mathsf{times}^*(x,x) \\
\mathsf{g}(x) \;&:=\; \mathsf{times}^*\left(\frac{7}{9},\, \mathsf{sq}^*(-\mathsf{sq}^*(-x))\right) \\
\mathsf{h}(x) \;&:=\; \underset{i}{\mathsf{M}}\; \mathsf{g}^{3(i+1)}(x) \\
\mathsf{H}(x) \;&:=\; \underset{i}{\mathsf{M}}\; (\mathsf{g}^{3(i+1)}(x))^2
\end{aligned}$$

Here times^* and sq^* are so named because they encode multiplication and square if the endpoints of the interval are renamed from $[-1,1]$ to $[0,1]$. Keeping to our convention that the interval is $[-1,1]$ the function g defined by g above is

$$g(x) \;=\; \frac{1}{9}x^4 - \frac{4}{9}x^3 - \frac{2}{9}x^2 + \frac{4}{3}x$$

which satisfies $g(0)=0$ and $g'(0)=\frac{4}{3}$. Hence, $(g^n)'(0) = (\frac{4}{3})^n$. This leads to the result below.

Proposition 2. *1. The function defined by* h *has derivative* ∞ *at* 0.

2. The function defined by H *has derivative* ∞ *when* 0 *is approached from above, and derivative* $-\infty$ *when* 0 *is approached from below.*

Since I is an abstract datatype, to compute with system I terms, we must give the datatype an implementation. In Sect. 2, we have implicitly discussed one such implementation in System T: the type $\mathsf{N} \to \mathsf{N}$ implements I, and System T versions of the programs in Fig. 1 implement the functions in the interface. Given this implementation, Prop. 3 below is immediate. We say that a function $f\colon [-1,1]^k \to [-1,1]$ is *I-definable* if there exists a closed System I term $u\colon \mathsf{I}^k \to \mathsf{I}$ such that $[\![u]\!] = f$.

Proposition 3. *Every I-definable function is T-representable.*

The converse, however, does not hold. We use square brackets for the truncation function $[\,\cdot\,] : \mathbb{R} \to [-1, 1]$ defined by:

$$[x] \;:=\; \min(1,\, \max(-1,\, x)) \enspace .$$

We write dbl for the function $x \mapsto [2x] : [-1, 1] \to [-1, 1]$.

Proposition 4. *The function* dbl *is T-representable but not I-definable.*

The non-definability of dbl shows that System I is, as already hinted above, limited in its capacity for rescaling the interval.

We end the section with the proof of Prop. 4. Using the notation of Fig. 1, a Haskell program computing dbl is:

```
dbl :: I -> I                     dbl (0:x) = x
dbl (1:1:x) = one                 dbl ((-1):(-1):x) = minusOne
dbl (1:0:x) = 1:(dbl (1:x))       dbl ((-1):0:x) = (-1):(dbl ((-1):x))
dbl (1:(-1):x) = 1:x              dbl ((-1):1:x) = (-1):x
```

This is easily converted into a System T term, showing that dbl is T-representable.

The non-definability proof uses logical relations. For every type τ we define a binary relation $\Delta_\tau \subseteq [\![\tau]\!] \times [\![\tau]\!]$ by:

$$\begin{aligned}
\Delta_{\mathsf{N}}(m, n) &\iff m = n \\
\Delta_{\mathsf{I}}(x, y) &\iff \text{if } x \in \{-1, 1\} \text{ or } y \in \{-1, 1\} \text{ then } x = y \\
\Delta_{\sigma\to\tau}(f, g) &\iff \forall x, y \in [\![\sigma]\!].\ \Delta_\sigma(x, y) \text{ implies } \Delta_\tau(f(x), g(y)) \\
\Delta_{\sigma\times\tau}((x, x'), (y, y')) &\iff \Delta_\sigma(x, y) \text{ and } \Delta_\tau(x', y')
\end{aligned}$$

Lemma 1. *For every System I constant $c\colon \tau$, it holds that $\Delta_\tau([\![c]\!], [\![c]\!])$.*

Proof. We consider two cases. To show that $\Delta_{(\mathsf{N}\to\mathsf{I})\to\mathsf{I}}(\mathrm{M}, \mathrm{M})$, suppose $\Delta_{\mathsf{N}\to\mathsf{I}}(f, f')$. Then, for all n, we have $\Delta_{\mathsf{I}}(f(n), f'(n))$. We must show that if $\mathrm{M}(f) \in \{-1, 1\}$ or $\mathrm{M}(f') \in \{-1, 1\}$ then $\mathrm{M}(f) = \mathrm{M}(f')$. We consider just the case that $\mathrm{M}(f) = -1$ (the others are similar). If $\mathrm{M}(f) = -1$ then $f(n) = -1$, for all $n \geq 0$. Since $\Delta_{\mathsf{I}}(f(n), f'(n))$, we have $f'(n) = -1$, for all $n \geq 0$. Thus $\mathrm{M}(f') = -1 = \mathrm{M}(f)$. We have thus shown that $\Delta_{\mathsf{I}}(\mathrm{M}(f), \mathrm{M}(f'))$ as required.

To show that $\Delta_{\mathsf{I}\to\mathsf{I}\to\mathsf{I}\to\mathsf{I}}(\mathrm{aff}, \mathrm{aff})$, suppose

$$\Delta_{\mathsf{I}}(x, x') \quad \text{and} \quad \Delta_{\mathsf{I}}(y, y') \quad \text{and} \quad \Delta_{\mathsf{I}}(z, z') \enspace . \tag{1}$$

We must show that if $\mathrm{aff}\, x\, y\, z \in \{-1, 1\}$ or $\mathrm{aff}\, x'\, y'\, z' \in \{-1, 1\}$ then $\mathrm{aff}\, x\, y\, z = \mathrm{aff}\, x'\, y'\, z'$. Suppose, without loss of generality, that $\mathrm{aff}\, x\, y\, z = -1$, i.e., $((1 - z)\, x + (1 + z)\, y)/2 = -1$. Then there are three possible cases: (i) $x = z = -1$; (ii) $y = -1$ and $z = 1$; (iii) $x = y = -1$. In each case, by (1), the corresponding equations hold for x', y', z'. Thus indeed $\mathrm{aff}\, x'\, y'\, z' = -1 = \mathrm{aff}\, x\, y\, z$. □

Lemma 2. *For every closed System I term $t\colon \tau$, it holds that $\Delta_\tau([\![t]\!], [\![t]\!])$.*

Proof. This is an immediate consequence of the previous lemma, by the fundamental lemma of logical relations. □

Proposition 5. *If $f\colon [-1,1] \to [-1,1]$ is I-definable and $f(x) \in \{-1,1\}$ for some $x \in (-1,1)$ then f is a constant function.*

Proof. Let $x \in (-1,1)$ be such that $f(x) \in \{-1,1\}$. Consider any $y \in (-1,1)$. Then $\Delta_{\mathsf{I}}(x,y)$. By Lemma 2, $\Delta_{\mathsf{I}\to\mathsf{I}}(f,f)$. Thus $\Delta_{\mathsf{I}}(f(x),f(y))$, whence $f(x) = f(y)$. Thus f is constant on $(-1,1)$, hence on $[-1,1]$ since continuous. □

The non-definability statement of Proposition 4 is an immediate consequence, as are many other non-definability results. For example, $\cos(x)$ and $\cos(\frac{x}{2})$ are not I-definable, even though $\frac{1}{2}\cos(\frac{x}{2})$ is (see above).

4 System II

We address the weakness identified above in the obvious way. System II ("double I") is obtained by adding dbl to System I.

$$\mathsf{dbl} \ : \ \mathsf{I} \to \mathsf{I} \qquad\qquad [\![\mathsf{dbl}]\!] \ = \ \mathrm{dbl} \ .$$

The equations from Fig. 2 are then augmented with:

(d) Equations for dbl:

$$\Gamma \vdash \mathsf{dbl}(\mathsf{m}(1,\, \mathsf{m}(1,\, t))) = 1 : \mathsf{I} \qquad \Gamma \vdash \mathsf{dbl}(\mathsf{m}(-1,\, \mathsf{m}(-1,\, t))) = -1 : \mathsf{I}$$
$$\Gamma \vdash \mathsf{dbl}(\mathsf{m}(0,\, t)) = t : \mathsf{I} \ .$$

Proposition 6. *1. Using (m), (a) and (E) only,* dbl *is the unique (up to provable equality) term of type* $\mathsf{I} \to \mathsf{I}$ *for which equations (d) hold.*
2. Using (m), (a) and (d) only, cancellation (C) is a consequence.

Using dbl, we can define, in System II, several useful functions (using the square bracket truncation notation from Sect. 3).

$$\begin{aligned}
[x+y] &:= \mathsf{dbl}(x \oplus y) & \mathsf{max}(0,x) &:= [[x-1]+1] \\
x \ominus y &:= x \oplus (-y) & \mathsf{max}(x,y) &:= \mathsf{dbl}\left(\left[\frac{x}{2} + \mathsf{max}\,(0, y \ominus x)\right]\right) \\
[x-y] &:= \mathsf{dbl}(x \ominus y) & \mathsf{min}(x,y) &:= -\mathsf{max}(-x,-y) \\
& & |x| &:= \mathsf{max}(-x,x)
\end{aligned}$$

Question 1. Are $\mathsf{max}(0,x)$, $\mathsf{max}(x,y)$ and $|x|$ definable in System I?

(The logical relation used in the proof of Prop. 4 does not help here.)

Having defined truncated versions of arithmetic functions, a very useful way of combining functions is by taking limits of Cauchy sequences. For *fast* Cauchy sequences (see Sect. 2), a limit-finding function $\mathsf{fastlim}\colon (\mathsf{N} \to \mathsf{I}) \to \mathsf{I}$ is definable:

$$\mathsf{fastlim} \ := \ \lambda f\!:\!\mathsf{N} \to \mathsf{I}.\ \mathsf{dbl}\left(\underset{n}{\mathsf{M}}\ \mathsf{dbl}^{n+1}(f(n+1) \ominus f(n))\right) \ .$$

We write fastlim for the function $(\mathbb{N} \to [-1,1]) \to [-1,1]$ defined by $\mathsf{fastlim}$.

Lemma 3. *Let $(x_n)_n$ be a sequence from $[-1,1]$. If $|x_{n+1}-x_n| \le 2^{-n}$, for all n, then* $\mathrm{fastlim}(n \mapsto x_n)$ *is the limit of the (fast) Cauchy sequence $(x_n)_n$.*

Of course, $\mathrm{fastlim}(n \mapsto x_n)$ always returns a value, even if $(x_n)_n$ is non-convergent. Also if $(x_n)_n$ converges, but too slowly, then $\mathrm{fastlim}(n \mapsto x_n)$ need not be the limit value.

We have seen that dbl is representable in System T. Thus the System T implementation of System I extends to an implementation of System II. Naturally, we say that $f\colon [-1,1]^k \to [-1,1]$ is *II-definable* if there exists a closed System II term $u\colon \mathsf{I}^k \to \mathsf{I}$ such that $[\![u]\!] = f$. Proposition 7 below is immediate.

Proposition 7. *Every II-definable function is T-representable.*

The main result of the paper is the converse.

Theorem 1. *Every T-representable function is II-definable.*

The rest of this section is devoted to the proof of Theorem 1. We need some auxiliary definitions.

Define $\mathsf{glue}\colon (\mathsf{I} \to \mathsf{I})^2 \to \mathsf{I} \to \mathsf{I}$ by

$$\mathsf{glue} \;:=\; \lambda\, f\, g\, x.\, \mathsf{dbl}\left(\mathsf{dbl}\left(\left(f\left(\mathsf{dbl}\left[x+\frac{1}{2}\right]\right) \oplus g\left(\mathsf{dbl}\left[x-\frac{1}{2}\right]\right)\right) \ominus \frac{1}{2}\, f(1)\right)\right) \quad .$$

which, whenever $f(1) = g(-1)$, satisfies

$$\text{glue } f \; g \; x \;=\; \begin{cases} f(2x+1) & \text{if } -1 \le x \le 0 \\ g(2x-1) & \text{if } 0 \le x \le 1 \;. \end{cases}$$

Next, for every $k \ge 1$, we define a System II term:

$$\mathsf{appr}_k\colon \mathsf{N} \to ((\mathsf{N} \to \mathsf{N})^k \to \mathsf{I}) \to (\mathsf{I}^k \to \mathsf{I})$$

The base case appr_1 is defined by primitive recursion on N to satisfy:

$$\begin{aligned} \mathsf{appr}_1\; 0\; h \;&=\; \mathsf{aff}\;(h(\overline{-1}))\;(h(\overline{1})) \quad \text{(where } \overline{-1} \text{ and } \overline{1} \text{ represent } -1 \text{ and } 1\text{)} \\ \mathsf{appr}_1\;(n+1)\; h \;&=\; \mathsf{glue}\;(\mathsf{appr}_1\; n\; (\lambda x.\, h(x \oplus (-1))))\;(\mathsf{appr}_1\; n\; (\lambda x.\, h(x \oplus 1))) \;\;. \end{aligned}$$

Given appr_k, the term appr_{k+1} is given by

$$\mathsf{appr}_{k+1}\; n\; h\; x_0\; x_1\; \ldots\; x_k \;=\; \mathrm{appr}_1\; n\; (\lambda y_0.\, \mathsf{appr}_k\; n\; (h\; y_0)\; x_1\; \ldots\; x_k)\; x_0 \;\;.$$

Let appr_k be the denotation of appr_k. If $h\colon (\mathbb{N} \to \mathbb{N})^k \to [-1,1]$ is real-extensional then the application $\mathrm{appr}_k\; n\; h$ produces a piecewise multilinear approximation to the function h, with the argument types changed from $\mathbb{N} \to \mathbb{N}$ to $[-1,1]$.

More precisely, the $\mathrm{appr}_k\; n$ function uses k-tuples of values from the set

$$\mathbb{Q}_n \;:=\; \{q_n^i \mid 0 \le i \le 2^n\} \quad \text{where } q_n^i := \frac{i}{2^{n-1}} - 1$$

to form a lattice of $(2^n+1)^k$ rational partition points in $[-1,1]^k$. The application $\mathrm{appr}_k\ n\ h$ then results in a function $[-1,1]^k \to [-1,1]$ that agrees with h at the partition points, and is (separately) affine in each coordinate between partition points. It is also affine in the h argument. The lemma below formalises this.

Lemma 4. *If $h\colon (\mathbb{N}\to\mathbb{N})^k \to [-1,1]$ represents $f\colon [-1,1]^k \to [-1,1]$ then:*

1. *For all $r_0,\ldots,r_{k-1} \in \mathbb{Q}_n$ we have:*

$$\mathrm{appr}_k\ n\ h\ r_0\ \ldots\ r_{k-1}\ =\ f\ r_0\ \ldots\ r_{k-1}$$

2. *For $0 \le j < k$, $0 \le i < 2^n$, and $0 \le \lambda \le 1$*

$$\begin{aligned}\mathrm{appr}_k\ n\ h\ x_0\ \ldots\ x_{j-1}\ \left(\tfrac{i+\lambda}{2^{n-1}}-1\right)\ x_{j+1}\ \ldots\ x_{k-1}\ &=\\ (1-\lambda)\,\mathrm{appr}_k\ n\ h\ x_0\ \ldots\ x_{j-1}\ q_n^i\ x_{j+1}\ \ldots\ x_{k-1}&\\ +\,\lambda\,\mathrm{appr}_k\ n\ h\ x_0\ \ldots\ x_{j-1}\ q_n^{i+1}\ x_{j+1}\ \ldots\ x_{k-1}&\end{aligned}$$

Note that $\left(\frac{i+\lambda}{2^{n-1}}-1\right) = ((1-\lambda)\,q_n^i + \lambda\,q_n^{i+1})$.

Also, if $h_1,h_2\colon (\mathbb{N}\to\mathbb{N})^k \to [-1,1]$ are real-extensional then

3. *For $0 \le \lambda \le 1$, we have:*

$$\begin{aligned}&\mathrm{appr}_k\ n\ ((1-\lambda)h_1+\lambda h_2)\ x_0\ \ldots\ x_{k-1}\ =\\ &\quad(1-\lambda)\,\mathrm{appr}_k\ n\ h_1\ x_0\ \ldots\ x_{k-1} + \lambda\,\mathrm{appr}_k\ n\ h_2\ x_0\ \ldots\ x_{k-1}\end{aligned}$$

In fact, under the conditions of the lemma, $\lambda n.\,\mathrm{appr}_k\ n\ h$ is a sequence of functions $[-1,1]^k \to [-1,1]$ that converges pointwise, and hence uniformly, to h. All that remains to be done is to extract a fast-converging subsequence, since then h can be defined using the fastlim function. In order to get a handle on the rate of convergence, we exploit the following classic fact [14]. (For $\alpha\colon \mathbb{N}\to\mathbb{N}$, and $k \in \mathbb{N}$, we write $\alpha\restriction_k$ for the sequence $\alpha(0),\ldots,\alpha(k-1) \in \mathbb{N}^k$.)

Lemma 5 (Definable modulus of uniform continuity). *Suppose we have a closed System T term*

$$t\colon (\mathsf{N}\to\mathsf{N}) \to (\mathsf{N}\to\mathsf{N})$$

Then there exists a closed System T term

$$U_t\colon \mathsf{N}\to\mathsf{N}$$

satisfying: for all $e \ge 0$, and for all $\beta,\gamma\colon \mathbb{N} \to \{0,1,2\}$ such that $\beta\restriction_{U_t(e)} = \gamma\restriction_{U_t(e)}$, it holds that $[\![t]\!](\beta)\restriction_e = [\![t]\!](\gamma)\restriction_e$.

We now complete the proof of Theorem 1. Suppose $t\colon (\mathsf{N}\to\mathsf{N})^k \to \mathsf{N}\to\mathsf{N}$ is a closed term that T-represents $f\colon [-1,1]^k \to [-1,1]$. Let U_t be a uniform modulus for continuity for t on $\mathbb{N} \to \{0,1,2\}$, as given by Lemma 5. Let $g_n\colon [-1,1]^k \to [-1,1]$ be defined by:

$$\mathsf{appr}_k\,(U_t(n+1))\,(\mathsf{real}\circ t)\ :\ \mathsf{I}^k \to \mathsf{I}\ .$$

Then for all $x_1, \ldots, x_k \in [-1, 1]$

$$|f(x_1, \ldots, x_n) - g_n(x_1, \ldots, x_n)| \;\leq\; 2^{-n} \;\; .$$

Therefore the term below II-defines f (where real is the easily defined system I term of type $(\mathsf{N} \to \mathsf{N}) \to \mathsf{I}$ implementing the function real from Section 2).

$$\lambda\, x_0 \, \ldots \, x_{k-1}.\, \mathsf{fastlim}\, (\lambda n.\, \mathsf{appr}_k\, (U_t(n+1))\, (\mathrm{real} \circ t)\, x_0 \, \ldots \, x_{k-1}) \;\; .$$

5 Mantissa-exponent Representation

There are many ways of extending signed binary to represent the full real line. Typically, one represents real number by a pair $\langle \alpha, z \rangle$ where $\alpha \in \{-1, 0, 1\}^\omega$, is the signed binary representation of $\mathrm{real}(\alpha) \in [-1, 1]$ and $z \in \mathbb{Z}$. One natural option is for $\langle \alpha, z \rangle$ to represent the real number $z + \mathrm{real}(\alpha)$, thus treating z as an *offset*. Another is to use α as a *mantissa* and z as an *exponent*, giving the real number $2^z\, \mathrm{real}(\alpha)$. Again, both representations are intertranslatable.

In Systems I and II, a variation on such representations is available. Instead of using signed binary to represent a number in $[-1, 1]$, it is natural to use the type I itself. Thus we can encode real numbers in Systems I and II, using the type $\mathsf{I} \times \mathsf{Z}$, where we write Z as an alternative notation for N to emphasise that the type is being used to encode all (including negative) integers (and we shall adopt similar suggestive notation for manipulation of integers). Curiously, under this approach, even the most basic functions cannot be programmed using the offset representation, so we are forced to use mantissa-exponent. Thus a term $\langle t, u \rangle \colon \mathsf{I} \times \mathsf{Z}$, represents the real number $2^{[\![u]\!]}\, [\![t]\!]$, where we mildly abuse notation to give u an interpretation $[\![u]\!] \in \mathbb{Z}$. We call this representation *semi-extensional*, since it combines a continuous value t, which is extensional, with a discrete scaling u, which is intensional. Although representations of real numbers are not unique, the continuous part is determined once the scaling is fixed.

It is straightforward to extend our main definability result to a characterisation of functions on $\mathbb{R}$ definable in System II. We say that a function $f \colon \mathbb{R}^k \to \mathbb{R}$ is T-representable, if there exists a System T term

$$t \colon ((\mathsf{N} \to \mathsf{N}) \times \mathsf{Z})^k \to (\mathsf{N} \to \mathsf{N}) \times \mathsf{Z}$$

that computes f under mantissa-exponent representation. And we say that f is I (resp. II)-representable if there exists a System I (resp. II) term

$$t \colon (\mathsf{I} \times \mathsf{Z})^k \to \mathsf{I} \times \mathsf{Z}$$

that computes f under mantissa-exponent representation.

Theorem 2. *A function $f \colon \mathbb{R}^k \to \mathbb{R}$ is T-representable if and only if it is II-representable.*

This result is essentially just an N-indexed version of Theorem 1. We omit the proof for space reasons.

Curiously, we do not know whether dbl is necessary for Theorem 2.

Question 2. Is every II-representable function $f\colon \mathbb{R}^k \to \mathbb{R}$ also I-representable?

A positive answer may sound implausible. But we now show that surprisingly many functions on real numbers can be defined in System I. At the same time, we show that reasoning about equality between functions on $\mathbb{R}$ can be reduced to equational reasoning in System I.

Equivalence between representations is given by the smallest equivalence relation on $[-1,1] \times \mathbb{Z}$ satisfying

$$\langle x, m\rangle \;\sim\; \left\langle \frac{x}{2},\, m+1 \right\rangle \ .$$

Indeed, this equivalence relation is defined explicitly by

$$\langle x, m\rangle \;\sim\; \langle y, n\rangle \quad\Longleftrightarrow\quad \frac{x}{2^{\max(m,n)-m}} = \frac{y}{2^{\max(m,n)-n}} \ ,$$

where the right-hand-side is an equality expressible in System I.

Proposition 8. *The relation $\sim$ is provably an equivalence relation in System I.*

The intended formulation of the proposition is that the transitivity (symmetry and reflexivity being trivial) of $\sim$ is a derivable inference rule in System I. The proof makes essential use of cancellation (C) from Fig. 2.

The basic arithmetic operations on $\mathbb{R}$ are definable in System I.

$$\begin{aligned}
\mathbf{0} \;&:=\; \langle 0, 0\rangle \\
\mathbf{1} \;&:=\; \langle 1, 0\rangle \\
-\langle x, m\rangle \;&:=\; \langle -x, m\rangle \\
\langle x, m\rangle + \langle y, n\rangle \;&:=\; \left\langle \frac{x}{2^{\max(m,n)-m}} \oplus \frac{y}{2^{\max(m,n)-n}},\, \max(m,n)+1 \right\rangle \\
\langle x, m\rangle \times \langle y, n\rangle \;&:=\; \langle x\,y,\; m+n\rangle
\end{aligned}$$

It is provable in System I that the above operations respect $\sim$. (Once again, by this, we mean that the inference rule expressing this property is derivable.) Also, the usual equations for the arithmetic operations are provable (commutativity, associativity, distributivity, etc.).

Since every rational number is System I definable, it follows that polynomials with rational coefficients are I-representable. We now show that we can also define limits of fast Cauchy sequences, as long the Cauchy sequences come with a witness to their speed of convergence.

Suppose we have a sequence $(\mathbf{x}_i)_i$ given by $\mathbf{x}_{(-)} \,:\, \mathsf{N} \to \mathsf{I} \times \mathsf{Z}$, such that the inequalities $|\mathbf{x}_{i+1} - \mathbf{x}_i| \leq 2^{-(i+1)}$ are witnessed by $d_{(-)}\colon \mathsf{N} \to \mathsf{I}$ satisfying

$$\mathbf{x}_{i+1} - \mathbf{x}_i \;\sim\; \langle d_i,\, -(i+1)\rangle \ .$$

Then we define

$$\lim_i \mathbf{x}_i \;:=\; \mathbf{x}_0 + \langle \underset{i}{\mathsf{M}}\, d_i,\, 0\rangle \ .$$

Given the definability of rational polynomials and Cauchy limits, it is not implausible that a positive answer to Question 2 might be modelled on a constructive proof of the Stone-Weierstrass theorem. But this needs further investigation.

Another direction to explore is how much analysis can be developed using the mantissa-exponent representation of real numbers with the mantissa taken from our abstract datatype I. It would be interesting to explore this both using just the equational logic of Systems I and II, and also in the richer context of dependent type theory.

Acknowledgements. We would like to thank Jeremy Avigad, Ulrich Kohlenbach, Yitong Li, John Longley and the anonymous referees for helpful suggestions.

References

1. A. Bauer, M.H. Escardó, and A. Simpson. Comparing functional paradigms for exact real-number computation. volume 2380 of *Lect. Not. Comp. Sci.*, pages 488–500, 2002.
2. H.J. Boehm. Constructive real interpretation of numerical programs. *SIGPLAN Notices*, 22(7):214–221, 1987.
3. H.J. Boehm and R. Cartwright. Exact real arithmetic: Formulating real numbers as functions. In Turner. D., editor, *Research Topics in Functional Programming*, pages 43–64. Addison-Wesley, 1990.
4. A. Bove and P. Dybjer. Dependent types at work. *Proceedings of Language Engineering and Rigorous Software Development, LNCS*, 5520:57–99, 2009.
5. L.E.J. Brouwer. Besitzt jede reelle Zahl eine Dezimalbruchentwicklung? *Math. Ann.*, 83:201–210, 1920.
6. P. Di-Gianantonio. *A Functional Approach to Computability on Real Numbers.* PhD thesis, Università Degli Studi di Pisa, Dipartamento di Informatica, 1993.
7. P. Di-Gianantonio. Real number computability and domain theory. *Information and Computation*, 127(1):11–25, 1996.
8. A. Edalat and M.H. Escardó. Integration in Real PCF. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic In Computer Science*, pages 382–393, New Brunswick, New Jersey, USA, 1996.
9. A. Edalat and P. Di Gianantonio. A language for differentiable functions. In *Proceedings of FoSSaCS*, 2013.
10. M.H. Escardó. PCF extended with real numbers. *Theoret. Comput. Sci.*, 162(1):79–115, 1996.
11. M.H. Escardó and A. Simpson. A universal characterization of the closed Euclidean interval. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 115–128. IEEE Computer Society, 2001.
12. U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics.* Monographs in Mathematics. Springer, 2008.
13. The Coq development team. *The Coq proof assistant reference manual.* LogiCal Project, 2004. Version 8.0.
14. A. S. Troelstra. Some models for intuitionistic finite type arithmetic with fan functional. *J. Symbolic Logic*, 42(2):194–202, 1977.
15. K. Weihrauch. *Computable analysis.* Springer, 2000.
16. E. Wiedmer. Computing with infinite objects. *Theoret. Comput. Sci.*, 10:133–155, 1980.