Semantics of a Sequential Language for Exact Real-Number Computation

J. Raymundo Marcial-Romero [a,*,1],
Martín H. Escardó [a]

[a] *University of Birmingham, Birmingham B15 2TT, England*

Abstract

We study a programming language with a built-in ground type for real numbers. In order for the language to be sufficiently expressive but still sequential, we consider a construction proposed by Boehm and Cartwright. The non-deterministic nature of the construction suggests the use of powerdomains in order to obtain a denotational semantics for the language. We show that the construction cannot be modelled by the Plotkin or Smyth powerdomains, but that the Hoare powerdomain gives a computationally adequate semantics. As is well known, Hoare semantics can be used in order to establish *partial* correctness only. Since computations on the reals are infinite, one cannot decompose total correctness into the conjunction of partial correctness and termination as it is traditionally done. We instead introduce a suitable operational notion of strong convergence and show that total correctness can be proved by establishing partial correctness (using denotational methods) and strong convergence (using operational methods). We illustrate the technique with a representative example.

Key words: exact real-number computation, sequential computation, semantics, non-determinism, PCF.

1 Introduction

This is a contribution to the problem of sequential computation with real numbers, where real numbers are taken in the sense of constructive mathematics [2]. It is fair

[*] Corresponding author.
Email addresses: jrm@cs.bham.ac.uk (J. Raymundo Marcial-Romero), mhe@cs.bham.ac.uk (Martín H. Escardó).

[1] Present address: División de Computación, UAEM, Ciudad Universitaria S/N, 50040, Toluca, Estado de México, México

to say that the computability issues are well understood [35]. Here we focus on the issue of designing programming languages with a built-in, abstract data type of real numbers. Recent research, discussed below, has shown that it is notoriously difficult to obtain sufficiently expressive languages with sequential operational semantics and corresponding denotational semantics which articulate the data-abstraction requirement. Based on ideas arising from constructive mathematics, Boehm and Cartwright [3], however, proposed a compelling operational solution to the problem. Yet, their proposal falls short of providing a full solution to the data abstraction problem, as it is not immediately clear what the corresponding denotational interpretation would be. A partially successful attempt at solving this problem has been developed by Potts [29] and Edalat, Potts and Sünderhauf [6], as discussed below.

In light of the above, the purpose of this paper is two-fold: (1) to establish the intrinsic difficulties of providing a denotational model of Boehm and Cartwright's operational approach, and (2) to show how it is possible to cope with the difficulties. Before elaborating on this research programme, we pause to discuss previous work.

Di Gianantonio [14], Escardó [11], and Potts et al. [28] have introduced various extensions of the programming language PCF with a ground type for real numbers. Each of these authors interprets the real numbers type as a variation of the interval domain introduced by Scott [30]. In the presence of a certain parallel conditional [26], all computable first-order functions on the reals are definable in the languages [14,8]. By further adding Plotkin's parallel existential quantifier [26], all computable functions of all orders become definable in the languages [14,7,10]. In the absence of the parallel existential quantifier, the expressivity of the languages at second-order types and beyond is not known. Partial results in this direction are developed by Normann [24].

It is natural to ask whether the presence of such parallel constructs is an artifact of the languages or whether they are needed for intrinsic reasons. Escardó, Hofmann and Streicher [9] have shown that, in the interval domain models, the parallelism is in fact unavoidable: weak parallel-or is definable from addition and other manifestly sequential unary functions, which indicates that addition, in these models, is an intrinsically parallel operation. Moreover, Farjudian [12] has shown that if the parallel conditional is removed from the language, only piecewise affine functions on the reals are definable.

Essentially, the problem is as follows. Because computable functions on the reals are continuous (see e.g. [35]), and because the real line is a connected space, any computable boolean-valued function on the reals is constantly true or constantly false unless it diverges for some inputs. Hence, definitions using the sequential conditional produce either constant total functions or partial functions. If one allows the boolean-valued functions to diverge at some inputs, then non-trivial predicates are obtained, and this, together with the parallel conditional, allow us to define the non-trivial total functions [11].

This phenomenon had been anticipated by Boehm and Cartwright [3], who also proposed a solution to the problem. In this paper we develop the proposed solution and study its operational and denotational semantics. The idea is based on the following observations. In classical mathematics, the *trichotomy* law "$x < y$, $x = y$ or $x > y$" holds for any pair of real numbers x and y, but, as is well known, it fails in constructive (and in classical recursive) mathematics. However, the following alternative *cotransitivity* law holds in constructive settings: for any two numbers $a < b$ and any number x, at least one of the relations $a < x$ or $x < b$ holds. Equivalently, one has that $(-\infty, b) \cup (a, \infty) = \mathbb{R}$. Boehm and Cartwright's idea is to consider a language construct $\mathrm{rtest}_{a,b}$, for $a < b$ rational, such that:

(1) $\mathrm{rtest}_{a,b}(x)$ evaluates to true or to false for every real number x,
(2) $\mathrm{rtest}_{a,b}(x)$ may evaluate to true iff $x < b$, and
(3) $\mathrm{rtest}_{a,b}(x)$ may evaluate to false iff $a < x$.

It is important here that evaluation never diverges for a convergent input. If the real number x happens to be in the interval (a, b), then the specification of $\mathrm{rtest}_{a,b}(x)$ allows it to evaluate to true or alternatively to false. The particular choice will depend on the particular implementation of the real number x and of the construct $\mathrm{rtest}_{a,b}$ (cf. [20]), and is thus determined by the operational semantics.

As application of the construction, we give an example of a recursive definition of a *sequential* program for addition, which is single-valued at total inputs, as required, but multi-valued at partial inputs. Thus, by allowing the output to be multi-valued at partial inputs, we are able to overcome the negative results of Escardó, Hofmann and Streicher mentioned above.

We take the view that the denotational value of $\mathrm{rtest}_{a,b}(x)$ lives in a suitable powerdomain of the booleans. Thus (1) if $a < x < b$ then the denotational value would be the set $\{\mathsf{true}, \mathsf{false}\}$, (2) if $a \not< x$ and $x < b$ then it would be the set $\{\mathsf{true}\}$, and (3) if $a < x$ and $x \not< b$ then it would be the set $\{\mathsf{false}\}$. Technically, one has to be careful regarding which subsets of the powerset are allowed, but this is tackled later in the body of the paper. One of our main results is that the Hoare powerdomain gives a computationally adequate denotational semantics. We also show that the Plotkin and Smyth powerdomains do not render the rtest construction continuous and hence cannot be used as models. These and other examples of powerdomains are discussed in the body of the paper.

As is well known, Hoare semantics can be used in order to establish *partial* correctness only. Because computations on the reals are infinite, one cannot decompose total correctness into the conjunction of partial correctness and termination, as is usually done for discrete data types. Instead, we introduce a suitable operational notion of strong convergence and show that total correctness can be proved by establishing partial correctness (using denotational methods) and strong convergence (using operational methods). The technique is illustrated by a proof of total correctness of our sequential program for addition. Further applications are discussed in the concluding section.

1.1 Related work.

Potts [29] considers a redundant if operator (rif) for his programming language LAR (an extension of PCF with linear fractional transformations), defined as

$$rif : \mathcal{I}^C K \times \mathcal{I}^C F^2 \times (\mathcal{I}^C K \to t)^2 \to t$$

$$rif \;\; x \;\; < \;\; (I, J);\; then \;\; f \;\; else \;\; g = \begin{cases} f(x), & \text{if } I \ll x; \\ g(x), & \text{if } J \ll x. \end{cases}$$

where $K \in \mathcal{I}^C \mathcal{R}^\infty$ and F is a dense subset of K. He uses the Hoare powerdomain to develop a denotational semantics for his language and prove computational adequacy. Our work justifies this choice. Potts considers a deterministic one-step reduction relation, while we consider a non-deterministic relation so as to have a precise match as possible with the denotational semantics in the case of multi-valued terms.

Edalat, Potts and Sünderhauf [6] had previously considered the denotational counterpart of Boehm and Cartwright's operational solution. However, they restrict attention to what can be referred to as single-valued, total computations. In particular, their computational adequacy result for their denotational semantics is restricted to this special case. Although it is indeed natural to regard this case as the relevant one, we have already met compelling examples, such as the fundamental operation of addition, in which sequentiality cannot be achieved unless one allows, for example, multi-valued outputs at partial inputs.

For their denotational semantics, they consider the Smyth powerdomain of a topological space of real numbers (which they refer to as the upper powerspace). Thus, they consider possibly non-deterministic computations of total real numbers, restricting their attention to those which happen to be deterministic. In the work reported here, we instead consider non-deterministic computations of total and partial real numbers. In other words, instead of considering a powerdomain of a space of real numbers, we consider a powerdomain of a domain of partial real numbers. Our computational adequacy result holds for general computations, total or partial, and whether deterministic or not. For our domain of partial real numbers, we consider the interval domain proposed by Scott [30], but the present findings are expected to apply to many possible notions of domain of partial real numbers.

Farjudian [13] has developed a programming language, which he called SHRAD, which satisfies the three requirements mentioned at the beginning of the paper: sequentiality, data abstraction and expressivity. In his work, he defines a sequential language in which all computable first order functions are definable. However extensionality is traded off for sequentiality, in the sense that all computable first order functions are extensional over total real numbers but not over partial real numbers. Hence functions such as the rounding functions, which are frequently used in practice, cannot be defined in SHRAD.

Di Gianantonio [15] also discusses the problem of sequential real-number computation in the presence of data abstraction, with some interesting negative results and translations of parallel languages into sequential ones.

In order to characterize computable functions on the real numbers, Brattka [4] introduces a class of relations that includes a contruction which is essentially the same as Boehm and Cartwright's multi-valued test discussed above. The main difference is that we articulate relations as functions with values on a powerdomain. With this, we are able to capture higher-type computation. Moreover, as discussed above, we take a powerdomain of the interval domain, not of the real line, and hence we are able to distinguish partiality from multi-valuedness: an interval gives a partially specified real number, and a set of intervals collects the possible (total or partial) outputs of a non-deterministic computation.

1.2 Organization.

Section 2 presents a running example that motivates the technical development that follows. Section 3 introduces some background. Section 4 studies the rtest construction from the point of view of powerdomains. Section 5 develops a programming language with the rtest construction and establishes computational adequacy for the denotational semantics developed in Section 4. Section 6 applies this to develop techniques for correctness proofs and gives sample applications. Section 7 summarizes the main results and discusses open problems and further work.

2 Running example

In order to motivate the use of the multi-valued construction discussed in the introduction, we give an example showing how it can be used to avoid the parallel constructions used in previous works on real-number computation. We take the opportunity to introduce some basic concepts and constructions studied in the technical development that follows.

In the programming language considered in [11], the average operation

$$(- \oplus -) \colon [0,1] \times [0,1] \to [0,1]$$

defined by

$$x \oplus y = (x + y)/2$$

can be implemented as follows:

```
$x \oplus y$= pif $x < c$
      then pif $y < c$
          then cons$_L$(tail$_L(x) \oplus$ tail$_L(y)$)
          else cons$_C$(tail$_L(x) \oplus$ tail$_R(y)$)
      else pif $y < c$
          then cons$_C$(tail$_R(x) \oplus$ tail$_L(y)$)
          else cons$_R$(tail$_R(x) \oplus$ tail$_R(y)$).
```

Here

$$c = 1/2, \quad L = [0, c], \quad C = [1/4, 3/4], \quad R = [c, 1],$$

the function $\mathtt{cons}_a \colon [0, 1] \to [0, 1]$ is the unique increasing affine map with image the interval a, i.e.,

$$\mathtt{cons}_L(x) = x/2, \qquad \mathtt{cons}_C(x) = x/2 + 1/4,$$
$$\mathtt{cons}_R(x) = x/2 + 1/2,$$

and the function $\mathtt{tail}_a \colon [0, 1] \to [0, 1]$ is a left inverse, i.e.

$$\mathtt{tail}_a(\mathtt{cons}_a(x)) = x.$$

More precisely, the following left inverse is taken, where κ_a is the length of a and μ_a is the left end-point of a:

$$\mathtt{tail}_a(x) = \max(0, \min(\kappa_a x + \mu_a, 1)).$$

Because equality on real numbers is undecidable, the relation $x < c$ is undefined (or diverges, or denotes $\perp$) if $x = c$. In order to compensate for this, one uses a *parallel conditional* such that

$$\mathtt{pif} \perp \mathtt{then}\ z\ \mathtt{else}\ z = z.$$

The intuition behind the above program is the following. If both x and y are in the interval L, then we know that $x \oplus y$ is in the interval L, if both x and y are in the interval R, then we know that $x \oplus y$ is in the interval R, and so on. The boundary cases are taken care of by the parallel conditional. For example, $1/2$ is both in L and R, and an unfolding of the program for $x = y = 1/2$ gives

```
$1/2 \oplus 1/2$ = pif $\bot$
           then pif $\bot$
                then cons$_L$(1 $\oplus$ 1)
                else cons$_C$(1 $\oplus$ 0)
           else pif $\bot$
                then cons$_C$(0 $\oplus$ 1)
                else cons$_R$(0 $\oplus$ 0).
```

All branches of the conditionals evaluate to $1/2$, but in an infinite number of steps. This can be seen as follows. A repeated unfolding of $1 \oplus 1$ gives the infinite expression $\texttt{cons}_R(\texttt{cons}_R(\texttt{cons}_R(\dots)))$. Denotationally speaking, the program computes the unique fixed point of $\texttt{cons}_R$, which is 1. Operationally speaking, the first unfolding says that the result of the computation, whatever it is, lives in the interval R, because, by definition, the image of $\texttt{cons}_R$ is R; the second unfolding says that the result is in the right half of the interval R, i.e. in the interval $[3/4, 1]$; the third unfolding tells us that the result is in the interval $[7/8, 1]$, and so on. Thus, the operational semantics applied to $1 \oplus 1$ produces a shrinking sequence of intervals converging to 1. The other cases are analogous.

Of course, a drawback of such a recursive definition is that, during evaluation, the number of parallel processes grows exponentially in the number of unfoldings. In order to overcome this, we switch back to the usual sequential conditional, and we replace the *partial* less-than test by the *multi-valued* test discussed in the introduction:

```
Average($x, y$)= if rtest$_{l,r}(x)$
                 then if rtest$_{l,r}(y)$
                      then cons$_L$(Average(tail$_L(x)$, tail$_L(y)$))
                      else cons$_C$(Average(tail$_L(x)$, tail$_R(y)$))
                 else if rtest$_{l,r}(y)$
                      then cons$_C$(Average(tail$_R(x)$, tail$_L(y)$))
                      else cons$_R$(Average(tail$_R(x)$, tail$_R(y)$)),
```

where c of the previous program splits into two points

$$l = 1/4, \qquad r = 3/4.$$

and this time we choose

$$L = [0, r], \quad C = [1/8, 7/8], \quad R = [l, 1].$$

The intuition behind this program is similar. What is interesting is that, despite the use of the multi-valued construction `rtest`, the overall result of the computation is single valued. In other words, different computation paths will give different shrinking sequences of intervals, but all of them will shrink to the same number. A proof of this fact and of correctness of the program is provided in Section 6, using the techniques developed below. For further examples see [22].

3 Background

For domain-theoretic concepts, the reader is referred to [1,27], and for topological concepts to [33,34] (see also [16]). Here we briefly summarize the notions and facts that are relevant to our purposes.

3.1 Continuous Domains

Let P be a set with a preorder $\sqsubseteq$. For a subset X of P and an element $x \in P$ we write

$$\begin{gathered} {\downarrow} X = \{y \in P \mid y \sqsubseteq x \text{ for some } x \text{ in } X\}, \\ {\uparrow} X = \{y \in P \mid x \sqsubseteq y \text{ for some } x \text{ in } X\}, \\ {\downarrow} x = {\downarrow}\{x\}, \qquad {\uparrow} x = {\uparrow}\{x\}. \end{gathered}$$

We also say that X is a *lower set* iff $X = {\downarrow} X$, and that X is an *upper set* iff $X = {\uparrow} X$.

Let x and y be elements of a directed complete partial order (dcpo) D. We say that x *is way-below* or *approximates* y, denoted $x \ll y$, if for every directed subset A of D, $y \sqsubseteq \bigsqcup A$ implies $\exists a \in A$ with $x \sqsubseteq a$. We say that x is *compact* if it approximates itself. We define $\twoheaduparrow x = \{y \in D \mid x \ll y\}$, $\twoheaddownarrow x = \{y \in D \mid y \ll x\}$ and $K(D) = \{x \in D \mid x \text{ is compact}\}$. We say that a subset B of a dcpo D is a *basis* for D, if for every element x of D the set $\twoheaddownarrow x \cap B$ contains a directed subset with supremum x. A dcpo is called a *continuous domain* or simply a domain if it has a basis. A dcpo is called an *algebraic domain* if it has a basis of compact elements. An example of an algebraic domain is the domain $\mathbb{T}_\perp = \{\perp, \mathsf{false}, \mathsf{true}\}$ of booleans, ordered by $\perp \sqsubseteq \mathsf{false}, \perp \sqsubseteq \mathsf{true}$. A function f from a domain D to a domain E is Scott continuos if it is monotone and $f(\bigsqcup A) = \bigsqcup f(A)$ for all directed subset A of D. A Scott closed subset of a domain D is a lower set closed under directed supremum. We say that a Scott closed set is finitely generated if it is the lower set of a finite set. The following is easily established:

Lemma 3.1 *If D is a continuous domain, C a finitely generated Scott closed subset of D and $f : D \to D$ Scott continuous then*

$$\downarrow\{f(x) \mid x \in C\} = \mathsf{cl}\{f(x) \mid x \in C\}.$$

where cl *denotes topological (Scott) closure.*

3.2 The Interval Domains $\mathcal{R}$ and $\mathcal{I}$

The set $\mathcal{R}$ of non-empty compact subintervals of the Euclidean real line ordered by reverse inclusion,

$x \sqsubseteq y$ iff $x \supseteq y$,

is a continuous domain, referred to as the *interval domain*. Here intervals are regarded as "partial numbers", with the singleton intervals playing the role of "total numbers". If we add a bottom element to $\mathcal{R}$, then $\mathcal{R}$ becomes a bounded complete continuous domain $\mathcal{R}_\perp$. For any interval $x \in \mathcal{R}$, we write

$\underline{x} = \inf\ x$ and $\overline{x} = \sup\ x$

so that $x = [\underline{x}, \overline{x}]$. Its length is defined by

$$\kappa_x = \overline{x} - \underline{x}.$$

A subset $A \subseteq \mathcal{R}$ has a least upper bound iff it has non-empty intersection, and in this case

$$\bigsqcup A = \bigcap A = \left[\sup_{a \in A} \underline{a}, \inf_{a \in A} \overline{a}\right].$$

The way-below relation of $\mathcal{R}$ is given by

$x \ll y$ iff $\underline{x} < \underline{y}$ and $\overline{y} < \overline{x}$.

A basis for $\mathcal{R}$ is given by the intervals with distinct rational (alternatively dyadic) end-points.

The set $\mathcal{I}$ of all non-empty closed intervals contained in the unit interval $[0, 1]$ is a bounded complete, countably based continuous domain, referred as the *unit interval domain*. The bottom element of $\mathcal{I}$ is the interval $[0, 1]$.

3.3 Powerdomains

Powerdomains [25,31,32] are usually constructed as ideal completions [18] of finite subsets of basis elements. For our purposes, it is more convenient to work with their topological representations [27,1,19], which we now summarize. It is enough for our purposes to restrict attention to ω-continuous dcpos, which we refer to as *domains* in this subsection.

A subset A of a dcpo D is called *Scott closed* if it is closed in the Scott topology, that is, if it is a lower set and is closed under the formation of suprema of directed subsets. We use the notation $\mathsf{cl}(A)$ for the topological closure of A, i.e. the smallest

Scott closed set containing A. A *lense* is a non-empty set that arises as the intersection of a Scott-closed set and a Scott compact upper subset. Here the notion of Scott compact set is to be understood in the topological sense (every cover consisting of Scott open sets has a finite subcover). On the set of lenses of a dcpo D, we define the *topological Egli-Milner ordering*, $\sqsubseteq_{\mathsf{TEM}}$ by $K \sqsubseteq_{\mathsf{TEM}} L$ if $L \subseteq \uparrow K$ and $K \subseteq \mathsf{cl}(L)$. Notice that in a finite domain such as the flat domain of booleans, the lenses are just order-convex sets, and that the topological Egli-Milner order coincides with the usual order-theoretical one [16]. This is because in a finite domain the closed sets are precisely the lower sets, and all sets are compact.

The *Plotkin powerdomain* $\mathcal{P}^P D$ of a domain D consists of the lenses of D under the Egli-Milner order, and the formal-union operation $A \uplus B$ is given by actual union $A \cup B$ followed by topological convex closure (intersection of all convex closed sets containing it). There is a natural topological embedding $\eta \colon D \to \mathcal{P}^P D$ given by $x \mapsto \{x\}$.

The *Smyth powerdomain* $\mathcal{P}^S D$ consists of the set of non-empty Scott-compact upper subsets ordered by *reverse* inclusion, with formal union given by actual union. In this case, we have a natural topological embedding $\eta \colon D \to \mathcal{P}^S D$ given by $x \mapsto \uparrow x$

The *Hoare powerdomain* $\mathcal{P}^H D$ consists of all non-empty Scott-closed subsets of D ordered by inclusion. Because we use this to obtain a denotational model of our language, we consider it in more detail. Least upper bounds are given by

$$\bigsqcup_{i \in I} A_i = \mathsf{cl} \bigcup_{i \in I} A_i.$$

The construction is the functor part of a monad, with action on continuous maps given by

$$\begin{aligned} \widehat{f} \colon\ \mathcal{P}^H D &\to \mathcal{P}^H E \\ A \quad &\mapsto \mathsf{cl} f[A] \end{aligned}$$

for any $f \colon D \to E$. Its unit is given by

$$\begin{aligned} \eta_D \colon\ D &\to \mathcal{P}^H D \\ x &\mapsto \ \downarrow x, \end{aligned}$$

which is also a topological embedding. Instead of considering multiplication, one can equivalently consider the extension operator [21, Proposition 2.14], in this case given by

$$\begin{aligned} \bar{f} \colon\ \mathcal{P}^H D \to &\quad \mathcal{P}^H E \\ A \quad \mapsto &\ \mathsf{cl} \textstyle\bigcup_{a \in A} fa \end{aligned}$$

for any continuous map $f \colon D \to \mathcal{P}^H E$. Finally, formal unions are given by actual unions as in the case of the Smyth powerdomain:

$$A \uplus B = A \cup B.$$

4 Semantics of the Multi-valued Construction

In order to make the development of the introduction precise, we assume that we are given a functorial powerdomain construction $\mathcal{P}$, in a suitable category of domains, with a natural embedding

$$\eta_D \colon D \to \mathcal{P}D$$

and a continuous formal-union operation

$$(- \uplus -) \colon \mathcal{P}D \times \mathcal{P}D \to \mathcal{P}D$$

for every domain D. Then the definition of the function $\mathrm{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$, where $a < b$ are real numbers, can be formulated as

$$\mathrm{rtest}_{a,b}(x) = \begin{cases} \eta(\mathsf{true}), & \text{if } x \in (-\infty, a], \\ \eta(\mathsf{true}) \uplus \eta(\mathsf{false}), & \text{if } x \in (a, b), \\ \eta(\mathsf{false}), & \text{if } x \in [b, \infty). \end{cases}$$

Because in our language there will be computations on the reals that diverge or fail to fully specify a real number, we need to embed the real line into a domain of total and partial real numbers. We choose to work with the domain $\mathcal{R}_\perp$, where $\mathcal{R}$ is the interval domain introduced in Section 3. Similarly, as usual, we enlarge the domain $\mathbb{T}$ of booleans with a bottom element. Hence we have to work with an extension $\mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ of the above function, which we denote by the same name:

$$\begin{tikzcd} \mathbb{R} \arrow[r, "\mathrm{rtest}_{a,b}"] \arrow[d] & \mathcal{P}\mathbb{T} \arrow[d] \\ \mathcal{R}_\perp \arrow[r, "\mathrm{rtest}_{a,b}"] & \mathcal{P}\mathbb{T}_\perp \end{tikzcd}$$

For the moment, we do not insist on any particular extension. However, in order for a powerdomain construction to qualify for a denotational model of the language, the minimum requirement is that it makes the $\mathrm{rtest}_{a,b}$ function continuous.

Lemma 4.1 *If* $\mathrm{rtest}_{a,b} \colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ *is a continuous extension of the function* $\mathrm{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$*, then the inequalities*

$$\eta(\mathsf{true}) \sqsubseteq \eta(\mathsf{true}) \uplus \eta(\mathsf{false}),$$
$$\eta(\mathsf{false}) \sqsubseteq \eta(\mathsf{true}) \uplus \eta(\mathsf{false})$$

must hold in the powerdomain $\mathcal{P}\mathbb{T}_\perp$

PROOF. Because the embedding $\mathbb{R} \hookrightarrow \mathcal{R}_\perp$ is continuous when $\mathbb{R}$ is endowed with its usual topology and $\mathcal{R}_\perp$ with its Scott topology, so is its composition with the function $\mathrm{rtest}_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$, which we denote by $r\colon \mathbb{R} \to \mathcal{P}\mathbb{T}_\perp$. (This is the diagonal of the above commutative square). In any dcpo, the relation $d \sqsubseteq e$ holds if and only if every neighbourhood of d is a neighbourhood of e. Let V be a neighbourhood of $t := \eta(\mathsf{true})$. We have to show that $n := \eta(\mathsf{true}) \uplus \eta(\mathsf{false}) \in V$. The set $U := r^{-1}(V)$ is open in $\mathbb{R}$ by continuity of $r : \mathbb{R} \to \mathcal{P}\mathbb{T}$. Because $r(a) = t \in V$, we have that $a \in r^{-1}(V) = U$. Hence, because U is open in $\mathbb{R}$, there is an open interval (u, v) with $a \in (u, v) \subseteq U$. Choose x such that $a < x < v$ and $x < b$, that is, such that $x \in (a, b) \cap (u, v) \subseteq U$. By construction, $r(x) = n$. But $x \in r^{-1}(V)$, which shows that $n \in V$ and hence that $t \sqsubseteq n$, which amounts to the first inequality. The second inequality is obtained in the same way. $\square$

Thus, any powerdomain not satisfying the above two inequalities does not qualify for a model. In particular, this rules out the Plotkin and Smyth powerdomains. In fact, for the Plotkin powerdomain one has that $\eta(\mathsf{true}) = \{\mathsf{true}\}$ and $\eta(\mathsf{false}) = \{\mathsf{false}\}$, and their formal union is $\{\mathsf{true}, \mathsf{false}\}$ because this set is order-convex, but the sets $\{\mathsf{true}\}$ and $\{\mathsf{true}, \mathsf{false}\}$ are incomparable in the Egli-Milner order. For the Smyth powerdomain, the same sets are obtained by the embedding, formal union is given by actual union, and hence the inequalities do not hold because the order is given by reverse inclusion. We omit routine proofs of the fact that e.g. the mixed [17] and the sandwich [5] powerdomains also fail to satisfy the inequalities and hence to make the $\mathrm{rtest}_{a,b}$ construction continuous.

On the other hand, for the Hoare powerdomain, the inequalities do hold. In fact, $\eta(\mathsf{true}) = \{\mathsf{true}, \perp\}$ and $\eta(\mathsf{false}) = \{\mathsf{false}, \perp\}$, their formal union is their actual union $\{\mathsf{true}, \mathsf{false}, \perp\}$, and the ordering is given by inclusion. Moreover:

Proposition 1 *There is a continuous extension* $\mathrm{rtest}^H_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ *of the function* $\mathrm{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$.

PROOF. The functions $f, g\colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ defined by

$$f(x) = \begin{cases} \eta(\mathsf{true}), & \text{if } x \subseteq (-\infty, b), \\ \perp, & \text{otherwise,} \end{cases}$$

$$g(x) = \begin{cases} \eta(\mathsf{false}), & \text{if } x \subseteq (a, \infty), \\ \perp, & \text{otherwise,} \end{cases}$$

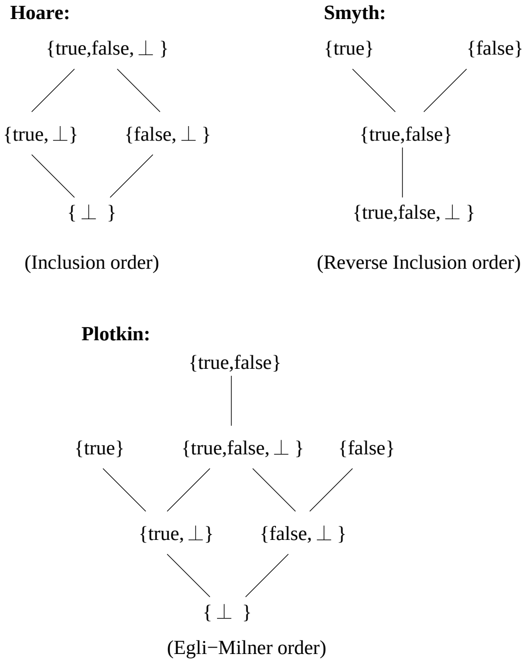

Fig. 1. Powerdomains of $\mathbb{T}_\perp$.

are easily seen to be continuous, and they are consistent because $\eta(\mathsf{true})$ and $\eta(\mathsf{false})$ are consistent elements. Hence their join

$$\mathrm{rtest}^H_{a,b} = f \sqcup g$$

is well-defined and continuous. An easy verification shows that this function has the required extension property. □

As we want to match our model with the operational semantics of the construction, it would be desirable to distinguish between the elements $\{\mathsf{true}\}$ and $\{\mathsf{true}, \perp\}$ in the model. However, the Hoare powerdomain does not distinguish them, and, on the other hand, as we have just seen, other powerdomains do not give a continuous interpretation of our construction. In order to overcome this problem when the Hoare powerdomain is used as a denotational model, one usually decomposes proofs of program correctness into partial correctness and termination. A related approach is considered in Section 6.

From now on, we denote $\mathrm{rtest}^H_{a,b} \colon \mathcal{R}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ simply by $\mathsf{rtest}_{a,b}$. In our applications, we are only interested in the situation $0 < a < b < 1$ and the restriction of this function to the domain $\mathcal{I}$ of closed subintervals of the interval $[0, 1]$, again written $\mathsf{rtest}_{a,b} \colon \mathcal{I} \to \mathcal{P}\mathbb{T}_\perp$.

4.0.0.1 Remark on the boundary cases of rtest.

Before proceeding to the main goal of this paper, we briefly digress to discuss a natural variation $\mathrm{rtest}'_{a,b}$:

$\mathbb{R} \to \mathcal{P}\mathbb{T}$ of the $\mathrm{rtest}_{a,b}$ construction, defined by

$$\mathrm{rtest}'_{a,b}(x) = \begin{cases} \eta(\mathsf{true}), & \text{if } x \in (-\infty, a), \\ \eta(\mathsf{true}) \uplus \eta(\mathsf{false}), & \text{if } x \in [a, b], \\ \eta(\mathsf{false}), & \text{if } x \in (b, \infty). \end{cases}$$

With a proof similar to that of Lemma 4.1, we conclude that if $\mathrm{rtest}'_{a,b}$ is continuous then

$$\eta(\mathsf{true}) \uplus \eta(\mathsf{false}) \sqsubseteq \eta(\mathsf{true})$$
$$\eta(\mathsf{true}) \uplus \eta(\mathsf{false}) \sqsubseteq \eta(\mathsf{false}).$$

This rules out the Plotkin and Hoare powerdomains, but not the Smyth powerdomain. However, it is not clear what the operational counterpart of this function would be. The function $\mathrm{rtest}_{a,b}$ is operationally computable because, for any argument x given intensionally as a shrinking sequence of intervals, the computational rules systematically establish one of the semidecidable conditions $a < x$ and $x < b$. However, the conditions $a \leq x$ and $x \leq b$ are not semi-decidable, and hence it is not immediately apparent what a computationally adequate operational semantics for rtest' would be. But it is interesting, as pointed out by one of the referees, that the cotransitivity law given in the introduction as a constructive justification of rtest can be equivalently formulated as "$a \leq x$ or $x \leq b$ whenever $a < b$". In any case, it is not clear to us, at the time of writing, whether or how this reformulation of the cotransitivity law would lead to a computational mechanism for rtest'.

5 A Programming Language for Sequential Real-Number Computation

We introduce the language LRT for the rtest construction, which amounts to the language considered by Escardó [11] with the parallel conditional removed and a constant for $\mathsf{rtest}_{a,b}$ added. We remark that this is a call-by-name language. Because real-number computations are infinite, and there are no canonical forms for partial real-number computations, it is not clear what a call-by-value operational semantics ought to be. We leave this as an open problem.

5.1 Syntax

The language LRT is an extension of PCF with a ground type for real numbers and suitable primitive functions for real-number computation. Its raw syntax is given by

$$\begin{aligned} &x \in \mathit{Variable}, \\ &t ::= \texttt{nat} \mid \texttt{bool} \mid \texttt{I} \mid t \to t, \\ &P ::= x \mid \texttt{n} \mid \texttt{true} \mid \texttt{false} \mid (+1)(P) \mid (-1)(P) \mid \\ &\qquad (=0)(P) \mid \texttt{if}\, P\, \texttt{then}\, P\, \texttt{else}\, P \mid \texttt{cons}_a(P) \mid \\ &\qquad \texttt{tail}_a(P) \mid \texttt{rtest}_{a,b}(P) \mid \lambda x : t.P \mid PP \mid \texttt{Y}P, \end{aligned}$$

where the subscripts of the constructs `cons`, `tail` are rational intervals and those of `rtest` are rational numbers. (We apologize for using the letters a and b to denote numbers and intervals in different contexts.) Terms of ground type `I` are intended to compute real numbers in the unit interval.

It is convenient for our purposes to first define the denotational and then the operational semantics.

5.2 *Denotational Semantics.*

The ground types `bool`, `nat` and `I` are interpreted as the Hoare powerdomain of the domains of booleans, natural numbers and intervals, respectively. Function types are interpreted as function spaces in the category of dcpos:

$$[\![\texttt{bool}]\!] = \mathcal{P}^H\mathbb{T}_\perp, \quad [\![\texttt{nat}]\!] = \mathcal{P}^H\mathbb{N}_\perp, \quad [\![\texttt{I}]\!] = \mathcal{P}^H\mathcal{I},$$

$$[\![\sigma \to \tau]\!] = [\![\sigma]\!] \to [\![\tau]\!].$$

This reflects the fact that we are considering a call-by-name language.

The interpretation of constants in LRT is defined as follows:

$$[\![\texttt{true}]\!] = \eta(\mathsf{true}), \quad [\![\texttt{false}]\!] = \eta(\mathsf{false}), \quad [\![\texttt{n}]\!] = \eta(\mathsf{n}),$$

$$[\![(+1)]\!] = \widehat{(+1)}, \quad [\![(-1)]\!] = \widehat{(-1)}, \quad [\![(=0)]\!] = \widehat{(=0)},$$

$$[\![\texttt{cons}_a]\!] = \widehat{\mathsf{cons}_a}, \quad [\![\texttt{tail}_a]\!] = \widehat{\mathsf{tail}_a},$$

$$[\![\texttt{rtest}_{a,b}]\!] = \overline{\mathsf{rtest}}_{a,b}, \qquad [\![\texttt{Y}]\!](F) = \bigsqcup_{n \geq 0} F^n(\perp),$$

$$[\![\texttt{if}]\!](B, X, Y) = \begin{cases} X, & \text{if } B = \eta(\mathsf{true}), \\ Y, & \text{if } B = \eta(\mathsf{false}), \\ X \uplus Y, & \text{if } B = \eta(\mathsf{true}) \uplus \eta(\mathsf{false}), \\ \perp, & \text{if } B = \perp. \end{cases}$$

Here the symbols η, $\widehat{\ }$, $\overline{\ }$ are defined as in Section 3.3, the functions $(+1), (-1), (=0)$ are the standard interpretations in the Scott model of PCF, the functions cons_a, tail_a are defined in Section 2, and the function $\mathsf{rtest}_{a,b}$ is defined in Section 4.

5.3 Operational Semantics

We consider a small-step style operational semantics for our language. We define the one-step reduction relation $\to$ to be the least relation containing the one-step reduction rules for evaluation of PCF [26] together with those given below.

We first need some preliminaries. For intervals a and b in $\mathcal{I}$, we define

$$ab = \mathsf{cons}_a(b),$$

where cons is the extension to the interval domain of the function defined in Section 2. This operation is associative, and has the bottom element of $\mathcal{I}$ as its neutral element [11]:

$$(ab)c = a(bc), \qquad a\bot = \bot a = a.$$

Moreover,

$$a \sqsubseteq b \iff \exists c \in \mathcal{I}.\ ac = b,$$

and this c is unique if a has non-zero length, i.e. it is not maximal, and in this case we denote c by

$$b \setminus a.$$

For intervals a and b, we define

$$a \leq b \iff \overline{a} \leq \underline{b}$$

and

$$a \uparrow b \iff \exists c.\ a \sqsubseteq c \text{ and } b \sqsubseteq c.$$

With this notation, the rules for Real PCF as defined in [11] are:

(1) $\mathtt{cons}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{ab} M$	
(2) $\mathtt{cons}_a M \rightarrow \mathtt{cons}_a M'$	if $M \rightarrow M'$ & (1) is not applicable
(3) $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathbf{Y}\mathtt{cons}_L$	if $b \leq a$
(4) $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathbf{Y}\mathtt{cons}_R$	if $b \geq a$
(5) $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{b\backslash a} M$	if $a \sqsubseteq b$ and $a \neq b$
(6) $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{(a \sqcup b)\backslash a}(\mathtt{tail}_{(a \sqcup b)\backslash b} M)$	if $a \uparrow b, a \not\sqsubseteq b, b \not\sqsubseteq a$, $b \not\leq a$ and $a \not\leq b$
(7) $\mathtt{tail}_a(M) \rightarrow \mathtt{tail}_a(M')$	if $M \rightarrow M'$ & (3)-(6) are not applicable
(8) $\mathtt{if}\ \mathtt{true}\ M\ N \rightarrow M$	
(9) $\mathtt{if}\ \mathtt{false}\ M\ N \rightarrow N$	
(10) $\mathtt{if}\ M\ N_1\ N_2 \rightarrow \mathtt{if}\ M'\ N_1\ N_2$	if $M \rightarrow M'$ & (8),(9) are not applicable

For our language LRT, we add:

(11) $\mathtt{rtest}_{b,c}(\mathtt{cons}_a\ M) \rightarrow \mathtt{true}$ if $\overline{a} < c$,
(12) $\mathtt{rtest}_{b,c}(\mathtt{cons}_a\ M) \rightarrow \mathtt{false}$ if $b < \underline{a}$,
(13) $\mathtt{rtest}_{b,c}\ M \rightarrow \mathtt{rtest}_{b,c}\ M'$ if $M \rightarrow M'$.

Remark 5.1

(1) Rule 1 plays a crucial role and amounts to the associativity law. The idea is that both a and b give partial information about a real number, and ab is the result of gluing the partial information together in an incremental way. See the paper [11] for a further discussion, including a geometrical interpretation.

(2) Notice that if the interval a is contained in the interval $[b, c]$, rules 11 and 12 can be applied.

(3) Rules 11-13 cannot be made deterministic given the particular computational adequacy formulation which is proved in Section 5.4. We shall show that the set of rewrite rules is rich enough to allow one to derive operationally everything that the denotational semantics suggests. This does not mean that we are giving a specification for an implementation of LRT. In the absense of $\mathtt{rtest}_{b,c}$, the rules 1-10 are deterministic without loss of computational adequacy. See Section 6 for a further discussion.

(4) In practice, one would like to avoid divergent computations by considering a strategy for application of the rules. This is the topic of Section 6 where we study total correctness. For the purposes of this section, we consider the non-deterministic view.

We now introduce a notion of operational meaning of a term, where the operational values are taken in a powerdomain too. The difference between this operational semantics and the denotational semantics given above is that the former is obtained by reduction but the latter is obtained, as usual, by compositional means.

Definition 5.2 *Firstly, we define the operational meaning of closed terms M of ground types γ in i steps of computation, written $[M]_i$, which is to be an element of the domain $[\![\gamma]\!]$.*

If $M : \mathtt{I}$, then we define

$$[M]_i = \uplus \, \{\eta(a) \mid \exists M' \exists k \le i, M \xrightarrow{k} \mathtt{cons}_a M'\}.$$

(If this set is empty, then of course $[M]_i = \bot$.) Here the relation $\xrightarrow{k}$ denotes the k-fold composition of the relation $\to$.

If $M : \mathtt{nat}$, then we define

$$[M]_i = \uplus \, \{\eta(n) \mid \exists k \le i, M \xrightarrow{k} \mathtt{n}\}$$

if this set is non-empty, and $[M]_i = \bot$ otherwise. The operational meaning of $M : \mathtt{bool}$ is defined similarly.

It is immediate that $[M]_i \sqsubseteq [M]_{i+1}$. Hence we can define

$$[M] = \bigsqcup_i [M]_i.$$

Of course, only in the case of the ground type of real numbers this definition is non-trivial, but it is convenient to have a uniform treatment for all types.

5.4 Computational Adequacy.

In our setting, *computational adequacy* amounts to the equation $[M] = [\![M]\!]$ for all closed terms M of ground type, where $[M]$ is the operational meaning of M and $[\![M]\!]$ is the denotational meaning of M defined above.

For a deterministic language such as PCF, soundness of the denotational semantics follows from the fact that $M \to N$ implies $[\![M]\!] = [\![N]\!]$. For our non-deterministic language, we rely on the following:

Lemma 5.3 $[\![M]\!] = \uplus \, \{[\![N]\!] \mid M \to N\}$ *(notice that this is a finite union).*

PROOF. The proof is by structural induction on M.

If M is a value, there is nothing to prove.

Suppose $M \equiv (-\mathtt{1})M'$ and $M \to N$, there are three rules that apply to predecessor.

First case: $M \equiv (-\mathtt{1})\mathtt{k}_{\mathtt{0}}$ and $(-\mathtt{1})\mathtt{k}_{\mathtt{0}} \to \mathtt{k}_{\mathtt{0}} \equiv N$,

$$\begin{aligned}\llbracket(-\mathtt{1})\mathtt{k}_{\mathtt{0}}\rrbracket &= \widehat{(-1)}\llbracket\mathtt{k}_{\mathtt{0}}\rrbracket = \widehat{(-1)}\{0,\bot\} = \mathsf{cl}\{(-1)0,(-1)\bot\}\\ &= \mathsf{cl}\{0,\bot\} = \{0,\bot\} = \llbracket\mathtt{k}_{\mathtt{0}}\rrbracket = \llbracket N\rrbracket.\end{aligned}$$

Second case: $M \equiv (-\mathtt{1})\mathtt{k}_{\mathtt{n+1}} \to \mathtt{k}_{\mathtt{n}} \equiv N$,

$$\begin{aligned}\llbracket(-\mathtt{1})\mathtt{k}_{\mathtt{n+1}}\rrbracket &= \widehat{(-1)}(\llbracket\mathtt{k}_{\mathtt{n+1}}\rrbracket) = \widehat{(-1)}\{n+1,\bot\} = \mathsf{cl}\{(-1)n+1,(-1)\bot\}\\ &= \mathsf{cl}\{n,\bot\} = \{n,\bot\} = \llbracket\mathtt{k}_{\mathtt{n}}\rrbracket = \llbracket N\rrbracket.\end{aligned}$$

Third case: $M \equiv (-\mathtt{1})M'$ and $M \to (-1)N'$ if $M' \to N'$. By the induction hypothesis, $\llbracket M'\rrbracket = \biguplus\,\{\llbracket N'\rrbracket \mid M' \to N'\}$, applying $\widehat{(-1)}$ to both sides of the equation:

$$\begin{aligned}\llbracket M\rrbracket = \llbracket(-1)M'\rrbracket &= \widehat{(-1)}\llbracket M'\rrbracket = \widehat{(-1)}\,(\biguplus\,\{\llbracket N'\rrbracket \mid M' \to N'\})\\ &= \biguplus\,\{\widehat{(-1)}\llbracket N'\rrbracket \mid M' \to N'\}\\ &= \biguplus\,\{\llbracket(-\mathtt{1})N'\rrbracket \mid M' \to N'\},\end{aligned}$$

as we wanted.

The proof for the other constants follows similarly, except for $\mathtt{rtest}_{a,b}$, whose proof we include below.

Suppose $M = \mathtt{rtest}_{p,q}(M')$. There are three possible cases:

First case: M is of the form $\mathtt{rtest}_{p,q}(M')$ where M' is not a $\mathtt{cons}_a$ term. Hence, the only single-step reductions available are of the form $M \to \mathtt{rtest}_{p,q}N'$ where $M' \to N'$. As the semantics of $\mathtt{rtest}_{p,q}$ is $\overline{\mathsf{rtest}}_{p,q}$, we get

$$\begin{aligned}\llbracket M\rrbracket &= \overline{\mathsf{rtest}}_{p,q}\,(\biguplus\,\{\llbracket N'\rrbracket \mid M' \to N'\})\\ &= \biguplus\,\{\overline{\mathsf{rtest}}_{p,q}\llbracket N'\rrbracket \mid M' \to N'\}\\ &= \biguplus\,\{\llbracket\mathtt{rtest}_{p,q}N'\rrbracket \mid M' \to N'\}\end{aligned}$$

Since the last expression exhausts the terms that are single-step derivable from M, we are done with this case.

Second case: M is of the form $\mathtt{rtest}_{p,q}(\mathtt{cons}_a(M''))$. Note that the above equality still holds but the last $\uplus$ does not exhaust the single-step derivations. Furthermore,

$$[\![M]\!] = \overline{\mathsf{rtest}}_{p,q}(\widehat{\mathsf{cons}}_a(M')) \sqsupseteq \mathtt{rtest}_{p,q}(a).$$

As $\uplus$ is inflationary, we can throw smaller terms into the above equation:

$$\begin{aligned}[\![M]\!] &= \uplus\, \{\mathtt{rtest}_{p,q}N' \mid M' \to N'\} \\ &= \mathsf{rtest}_{p,q}(a) \uplus \left(\biguplus \{[\![\mathtt{rtest}_{p,q}N']\!] \mid M' \to N'\}\right)\end{aligned}$$

Now $\mathsf{rtest}_{p,q}(a)$ is exactly the set

$$\biguplus \{[\![b]\!] \mid M \to b \text{ and } b \in \{\mathtt{true}, \mathtt{false}\}\}\,. \qquad \Box$$

Hence, by induction on the length j of the evaluation using the previous lemma, for every j, $[\![M]\!] = \uplus\, \{[\![N]\!] \mid M \xrightarrow{j} N\}$.

Lemma 5.4 (Soundness) *For all closed terms M of ground type,*

$$[M] \sqsubseteq [\![M]\!].$$

PROOF. It suffices to show that, for all closed terms M of ground type,

$$[M]_i \sqsubseteq [\![M]\!].$$

Let $b \in [M]_i, b \neq \bot$. By definition, $b \sqsubseteq a$ for some a and M' such that $M \xrightarrow{i} \mathtt{cons}_a M'$. Because $\widehat{\mathsf{cons}_a}[\![M']\!] = [\![\mathtt{cons}_a M']\!]$, Lemma 5.3 shows that $b \in {\downarrow}[\![\mathtt{cons}_a M']\!]$. Therefore $b \in [\![M]\!]$ because $a \sqsubseteq \mathsf{cons}_a(x)$ for all $x \in \mathcal{I}$, and in particular for all $x \in [\![M']\!]$. $\Box$

In order to establish completeness, we proceed as in [26,11].

Definition 5.5 We define a notion of computability for closed terms by induction on types as follows:

(1) A closed term M of ground type is computable whenever $[\![M]\!] \sqsubseteq [M]$,
(2) A closed term $M : \sigma \to \tau$ is computable whenever $MQ : \tau$ is computable for every closed computable term Q of type σ,

An open term $M : \sigma$ with free variables $x_1, \ldots, x_n$ of type $\sigma_1, \ldots, \sigma_n$ is computable whenever $[N_1/x_1] \cdots [N_n/x_n]M$ is computable for every family $N_i : \sigma_i$ of closed computable terms.

Because $\mathcal{P}^H(D)$ is a continuous domain if D is, we have:

Lemma 5.6 *A closed term M of ground type is computable iff for every $X \ll [\![M]\!]$ there is i with $X \sqsubseteq [M]_i$.*

PROOF. $(\Rightarrow)$ Suppose that M is computable and let $X \ll [\![M]\!]$. We have that $[M]_1 \sqsubseteq [M]_2 \sqsubseteq \cdots$ is a chain whose supremum is $[M]$, and hence there is i with $X \sqsubseteq [M]_i$. $(\Leftarrow)$ By continuity of the Hoare powerdomain of a continuous domain, in order to show that $[\![M]\!] \sqsubseteq [M]$, it suffices to show that for all $X \ll [\![M]\!]$, $X \sqsubseteq [M]$. But this holds by hypothesis. □

Recall the following from domain theory [1,16].

Lemma 5.7 *For any continuous function $f : D \to E$ of continuous dcpos, if $y \ll f(x)$ then there is $x' \ll x$ with $y \ll f(x')$.*

Lemma 5.8 (Completeness) *Every term is computable.*

PROOF. The proof is by structural induction on the formation rules of terms.

Constants: **(1)** $\mathtt{rtest}_{p,q}$ is computable:

We have to show that

$$[\![\mathtt{rtest}_{p,q}M]\!] \sqsubseteq [\mathtt{rtest}_{p,q}M]$$

for computable M. So

$$\begin{aligned}
[\![\mathtt{rtest}_{p,q}M]\!] &= \overline{\mathsf{rtest}}_{p,q}[\![M]\!] \\
&\sqsubseteq \overline{\mathsf{rtest}}_{p,q}[M] \\
&= \overline{\mathsf{rtest}}_{p,q} \bigsqcup_i [M]_i \\
&= \bigsqcup_i \overline{\mathsf{rtest}}_{p,q}[M]_i \\
&= \bigsqcup_i \overline{\mathsf{rtest}}_{p,q} \biguplus \left\{\eta(a) \mid \exists M' \exists k \leq i. M \to^k \mathtt{cons}_a M'\right\} \\
&= \bigsqcup_i \biguplus \left\{\overline{\mathsf{rtest}}_{p,q}(\eta(a)) \mid \exists M' \exists k \leq i. M \to^k \mathtt{cons}_a M'\right\} \\
&= \bigsqcup_i \biguplus \left\{\mathsf{rtest}_{p,q}(a) \mid \exists M' \exists k \leq i. M \to^k \mathtt{cons}_a M'\right\}.
\end{aligned}$$

But when $M \to^k \mathtt{cons}_a M'$ holds, so does $\mathtt{rtest}_{p,q}(a) \sqsubseteq [\mathtt{rtest}_{p,q}M]_{k+1} \sqsubseteq [\mathtt{rtest}_{p,q}M]$. So the directed sup of formal joins also lies below $[\mathtt{rtest}_{p,q}M]$.

(2) `if` is computable:

We have to show that

$$[\![\texttt{if}\ L\ M\ N]\!] \sqsubseteq [\texttt{if}\ L\ M\ N].$$

Suppose $\eta(\mathsf{true}) \sqsubseteq [\![L]\!]$. By the induction hypothesis, $[\![L]\!] \sqsubseteq [L]$, so $L \to^{l} \texttt{true}$ for some l. Thus $\texttt{if}\ L\ M\ N \to^{l+1} M$. Hence, $[\![M]\!] \sqsubseteq [\texttt{if}\ L\ M\ N]$. Similarly, if $\eta(\texttt{false}) \sqsubseteq [\![L]\!]$, then $[\![M]\!] \sqsubseteq [\texttt{if}\ L\ M\ N]$. Now, we need the four cases of the proof: if $[\![L]\!] = \eta(\bot)$, then $[\![\texttt{if}\ L\ M\ N]\!] = \eta(\bot)$; if $[\![L]\!] = \eta(\mathsf{true})$, then $[\![\texttt{if}\ L\ M\ N]\!] = [\![M]\!]$; if $[\![L]\!] = \eta(\mathsf{false})$, then $[\![\texttt{if}\ L\ M\ N]\!] = [\![N]\!]$; and if $[\![L]\!] = \eta(\mathsf{true}) \uplus \eta(\mathsf{false})$, then $[\![\texttt{if}\ L\ M\ N]\!] = [\![M]\!] \uplus [\![N]\!]$. Because $\uplus$ is inflationary (and $\eta(\bot)$ is the identity for it); in all four cases $[\![\texttt{if}\ L\ M\ N]\!] \sqsubseteq [\texttt{if}\ L\ M\ N]$.

(3) $\texttt{cons}_a$ is computable:

We have to show that if M is computable, then so is $\texttt{cons}_a M$.

Assume that $[\![\texttt{cons}_a M]\!] \neq \bot$ for a computable term M of type $\texttt{I}$. Let $Y \ll [\![\texttt{cons}_a M]\!] = \widehat{\mathsf{cons}}_a[\![M]\!]$. We need to show that there is i with $Y \sqsubseteq [\mathsf{cons}_a M]_i$. By Lemma 5.7, there is $X \ll [\![M]\!]$ with $Y \ll \widehat{\mathsf{cons}}_a X$. As M is computable, there is j such that $X \sqsubseteq [M]_j$. Because $Y \sqsubseteq \widehat{\mathsf{cons}}_a X$ and by monotonicity of $\widehat{\mathsf{cons}}_a$, we have that $Y \sqsubseteq \widehat{\mathsf{cons}}_a[M]_j$. So for every $y \in Y$, there is $m \in \widehat{\mathsf{cons}}_a[M]_j$, with $y \sqsubseteq m$. Let $m \in \widehat{\mathsf{cons}}_a[M]_j$, by Lemma 3.1 there is $t \in [M]_j$ with $m \sqsubseteq \mathsf{cons}_a(t) = at$. Because there is $t \in [M]_j$, we deduce that there is M' such that the reduction $M \xrightarrow{k} \texttt{cons}_t M',\ k \leq j$ holds, and so $\texttt{cons}_a M \xrightarrow{k} \texttt{cons}_a(\texttt{cons}_t M') \xrightarrow{1} \texttt{cons}_{at} M'$. Hence we can take $i = j + 1$.

(4) $\texttt{tail}_a$ is computable:

We have to show that if M is computable, then so is $\texttt{tail}_a M$. Assume that $[\![\texttt{tail}_a M]\!] \neq \bot$ for a computable term M of type $\texttt{I}$. Let $Y \ll [\![\texttt{tail}_a M]\!] = \widehat{\mathsf{tail}}_a[\![M]\!]$. We need to show that there is i with $Y \sqsubseteq [\mathsf{tail}_a M]_i$. By lemma 5.7, there is $X \ll [\![M]\!]$ with $Y \ll \widehat{\mathsf{tail}}_a X$. As M is computable, there is j such that $X \sqsubseteq [M]_j$. Because $Y \neq \{\bot\}$, it follows that $[M]_j \not\sqsubseteq \{a\}$ in the Egli–Milner order, and if $[M]_j \sqsubseteq \{a\}$ then $Y \ll \widehat{\mathsf{tail}}_a X \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j \sqsubseteq \widehat{\mathsf{tail}}_a\{a\} = \mathsf{cl}\{\bot\} = \{\bot\}$. Then exactly one of the following four cases holds:

(a) $[M]_j \leq \{a\}$: Then since $X \sqsubseteq [M]_j$, we have that $\widehat{\mathsf{tail}}_a X \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j$ and since $Y \sqsubseteq \widehat{\mathsf{tail}}_a X$, we have $Y \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j$. So for every $y \in Y$, there is $m \in \widehat{\mathsf{tail}}_a[M]_j$ with $y \sqsubseteq m$. Let $m \in \widehat{\mathsf{tail}}_a[M]_j$, so by lemma 3.1 there is $t \in [M]_j$ with $m \sqsubseteq \mathsf{tail}_a t$. Because there is $t \in [M]_j$ it follows that there is M' such that $M \xrightarrow{k} \texttt{cons}_t M', k \leq j$ holds. Because $[M]_j \leq \{a\}$ we conclude that $\texttt{tail}_a M \xrightarrow{k} \texttt{tail}_a(\texttt{cons}_t M') \xrightarrow{1} \texttt{Ycons}_L$. Hence we can take $i = k + 1$.

(b) $\{a\} \leq [M]_j$ Similar to 1.

(c) $\{a\} \sqsubseteq [M]_j$: Then since $X \sqsubseteq [M]_j$, we have that $\widehat{\mathsf{tail}}_a X \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j = \{b \setminus a \mid b \in [M]_j\}$ and since $Y \sqsubseteq \widehat{\mathsf{tail}}_a X$, we have that $Y \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j$. So for every $y \in Y$, there is $m \in \widehat{\mathsf{tail}}_a[M]_j$ with $y \sqsubseteq m$. Let $m \in \widehat{\mathsf{tail}}_a[M]_j$, so there is $t \in [M]_j$ with $m \sqsubseteq \mathsf{tail}_a t = t \setminus a$. Because there is $t \in [M]_j$ it follows that there is M' such that $M \xrightarrow{k} \mathtt{cons}_t M', k \leq j$ holds. We conclude that $\mathtt{tail}_a M \xrightarrow{k} \mathtt{tail}_a(\mathtt{cons}_t M') \xrightarrow{1} \mathtt{tail}_m M'$. Hence we can take $i = k + 1$.

(d) $\{a\} \uparrow [M]_j$: Then since $X \sqsubseteq [M]_j$, we have that $\widehat{\mathsf{tail}}_a X \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j = \{(a \sqcup b) \setminus a \mid b \in [M]_j\}$ and since $Y \sqsubseteq \widehat{\mathsf{tail}}_a X$, we have that $Y \sqsubseteq \mathsf{tail}_a[M]_j$. So for every $y \in Y$, there is $m \in \widehat{\mathsf{tail}}_a[M]_j$ with $y \sqsubseteq m$. Let $m \in \widehat{\mathsf{tail}}_a[M]_j$, so there is $t \in [M]_j$ with $m \sqsubseteq \mathsf{tail}_a t = (a \sqcup t) \setminus a$. Because there is $t \in [M]_j$ it follows that there is M' such that the reduction $M \xrightarrow{k} \mathtt{cons}_t M', k \leq j$ holds. We conclude that $\mathtt{tail}_a M \xrightarrow{k} \mathtt{tail}_a(\mathtt{cons}_t M') \xrightarrow{1} \mathtt{tail}_m M'$. Hence we can take $i = k + 1$.

(5) For $M \equiv (+\mathtt{1}), (-\mathtt{1}), (= \mathtt{0})$ the proof is similar to the `if` case.

(6) If M is computable so is $\lambda\alpha M$:

We must show that $LN_1, \ldots N_n$ is computable whenever $N_1, \ldots N_n$ are closed computable terms and L is a closed instantiation of $\lambda\alpha M$ by computable terms. Here L must have the form $\lambda\alpha M'$ where M' is an instantiation of all free variables of M, except α, by closed computable terms.

If $P \ll [\![LN_1 \ldots N_n]\!]$ then we have $P \ll [\![[N_1/\alpha]M'N_2 \ldots N_n]\!] = [\![LN_1 \ldots N_n]\!]$. But $[N_1/\alpha]M'$ is computable and so therefore $[N_1/\alpha]M'N_2 \ldots N_n$. Hence there is j with $P \sqsubseteq [[N_1/\alpha]M'N_2 \ldots N_n]_j$. Since $LN_1 \ldots N_n \to [N_1/\alpha]M'N_2 \ldots N_n$ and the reduction relation preserves meanings, in order to evaluate $LN_1 \ldots N_n$ it suffices to evaluate $[N_1/\alpha]M'N_2 \ldots N_n$. Hence we can take $i = j$.

(7) $\mathtt{Y}_\sigma$ is computable:

In order to prove that $\mathtt{Y}_\sigma$ is computable it suffices to show that the term

$$\mathtt{Y}_{(\sigma_1,\ldots,\sigma_k,\mathcal{P}I)} N_1 \cdots N_k$$

is computable whenever $N_1 : \sigma_1, \ldots, N_k : \sigma_k$ are closed computable terms. It follows from (6) above that the terms $\mathtt{Y}_\sigma^{(n)} := \lambda f.f^n(\perp)$ are computable, because the proof of computability of $\mathtt{Y}_\sigma^{(n)}$ depends only on the fact that variables are computable and that the combination and abstraction formation rules preserve computability.

Let $P \ll [\![\mathtt{Y}N_1 \cdots N_K]\!]$ be different from $\perp$. Because $[\![\mathtt{Y}]\!] = \bigsqcup[\![\mathtt{Y}^{(n)}]\!]$, by a basic property of the way-below relation of any continuous dcpo, there is some n such that $P \ll [\![\mathsf{Y}^{(n)}N_1 \cdots N_K]\!]$. Since $\mathsf{Y}^{(n)}$ is computable, there is j with $P \sqsubseteq [\mathsf{Y}^{(n)}N_1 \cdots N_k]_j$. Since there is a term M with $\mathtt{Y}^{(n)}N_1 \cdots N_k \xrightarrow{j} \mathtt{cons}_c M$. Using the *syntactic information order* (see [26,11]), and Lemma 5.9 below, $\mathsf{Y}^{(n)} \preccurlyeq \mathsf{Y}$ we have that $\mathtt{Y}N_1 \cdots N_k \xrightarrow{j} \mathtt{cons}_c M$ for some M and therefore $i = j$. □

As in the last part of the above proof, we denote the syntactic order by $\preccurlyeq$ (see [26] or [11]).

Lemma 5.9 *If $M \preccurlyeq N$ and $M \to M_1, M \to M_2, \cdots, M \to M_n$ then either $\forall i, M_i \preccurlyeq N, 1 \leq i \leq n$ or else for some terms $N_1, N_2, \ldots, N_m, N \to N_1, N \to N_2, \cdots, N \to N_m$, and $\forall M_i, \exists N_j, M_i \preccurlyeq N_j, 1 \leq i \leq n, 1 \leq j \leq m$*

PROOF. The case that we must consider is the one that involves $\mathtt{rtest}_{a,b}$. The other cases are treated as in Real PCF.

(1) $\mathtt{rtest}_{a,b}M \preccurlyeq \mathtt{rtest}_{a,b}M$ holds by definition.

(2) $M \equiv \mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M'' \equiv N$ and $M \to \mathtt{true}$. These conditions hold if $\mathtt{rtest}_{a,b}M \to \mathtt{rtest}_{a,b}(\mathtt{cons}_c M''')$ and $\overline{c} < b$. By the induction hypothesis, $M' \to M''$ so $\mathtt{rtest}_{a,b}M'' \to \mathtt{rtest}_{a,b}(\mathtt{cons}_d M^{iv})$ where $\overline{d} < b$ so $\mathtt{rtest}_{a,b}M'' \to \mathtt{true}$ and $\mathtt{true} \preccurlyeq \mathtt{true}$.

(3) $M \equiv \mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M'' \equiv N$ and $M \to \mathtt{false}$. Similar to the previous case.

(4) $M \equiv \mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M'' \equiv N$ and $M \to \mathtt{true}, M \to \mathtt{false}$. These follows if $\mathtt{rtest}_{a,b}M \to \mathtt{rtest}_{a,b}(\mathtt{cons}_c M''')$ and $a < c < b$. By the induction hypothesis, $M' \to M''$ so $\mathtt{rtest}_{a,b}M'' \to \mathtt{rtest}_{a,b}(\mathtt{cons}_d M^{iv})$ where $a < d < b$ so $\mathtt{rtest}_{a,b}M'' \to \mathtt{true}, \mathtt{rtest}_{a,b}M'' \to \mathtt{false}$ and $\mathtt{true} \preccurlyeq \mathtt{true}, \mathsf{false} \preccurlyeq \mathsf{false}$. □

In summary:

Theorem 5.10 *Computational adequacy holds; that is, for every closed term M of ground type, the operational and denotational meanings of M coincide:*

$$[M] = [\![M]\!].$$

6 Program Correctness

We now develop tools for establishing correctness of LRT programs. In order to show that a given program is correct with respect to a given specification, we show that

(1) if it converges, then it satisfies the specification, and
(2) it in fact converges.

In our examples, condition 1 will be achieved by applying the denotational semantics with the aid of computational adequacy, and condition 2 will be achieved using the operational semantics directly. Hence our first task is to define a suitable operational notion of convergence for terms of real-number type.

Firstly, notice that the operational semantics defined in Section 5.3 allows divergence when rule 13 for $\mathtt{rtest}_{a,b}$ is applied infinitely often. But the only purpose of this rule is to get a sufficiently precise approximation of the argument, so that rules 11 and/or 12 can be eventually applied, provided such an approximation exists. Hence we agree that

> *we do not apply rule 13 for $\mathtt{rtest}_{a,b}$ infinitely often unless rules 11-12 are never applicable.*

Definition 6.1 The subrelation of the reduction relation $\rightarrow$ that arises in this way will be denoted by $\Rightarrow$.

Secondly, in the case of a term of the form $\mathtt{rtest}_{a,b}(M)$, after finitely many applications of rule 13 to compute an approximation of the argument M, we will have three situations:

(1) Both rules 11 and 12 become applicable.
(2) One and only one of the rules 11 and 12 becomes applicable.
(3) It is still not possible to apply rules 11 and 12, and hence one should keep applying rule 13, getting better and better approximations of M, either
    (a) for ever, or
    (b) so that we eventually arrive at one of the previous situations (1) or (2), and the computation converges to a truth value.

If the situation (3a) may take place, we say that the term *may diverge*, and otherwise, that it *must converge*. If the situation (1) takes place, we may imagine that the computation bifurcates into two subcomputations, each of which will give an answer or diverge. For our definition of strong convergence, to be given below, we require that both converge. In practice, an implementation of the language will typically choose one of the branches, according to some strategy, which will not necessarily be known to the programmer, and such a branch will then lead to an answer or divergence. In this case, the programmer has to ensure that any possible answer satisfies the desired specification, or that both branches will in fact lead to the same answer (as will be the case with our running example).

In theory, if situation (2) takes place, one can carry on with the computation produced by the corresponding branch, and, at the same time, repeatedly apply rule 13 in parallel so that maybe the other rule becomes applicable too and one has two computations as in situation (1). This corresponds to the relation $\Rightarrow$ defined above.

In practice, we work with a deterministic, but unspecified strategy, as follows:

Definition 6.2 *A* strategy *is a subrelation $\Rrightarrow$ of $\Rightarrow$ such that*

(1) $\Rrightarrow$ is singled-valued, i.e. for any M there is at most one N such that $M \Rrightarrow N$,
(2) if there is an N such that $M \Rightarrow N$, then there is also an N such that $M \Rrightarrow N$.

Notice that the only reason the relation $\Rightarrow$ is multi-valued is the presence of rules 11 and 12. In summary, the relation $\Rightarrow$ removes inessential infinite computations from $\rightarrow$, and $\Rrightarrow$ gives a deterministic strategy for the application of $\rightarrow$.

$$(\Rrightarrow) \subseteq (\Rightarrow) \subseteq (\rightarrow).$$

Here are some examples of deterministic relations $\Rrightarrow$

(1) At each stage of the reduction of a term, apply the first applicable rule, for the ordering of the rules given in Section 5.3.
(2) The same strategy as 1, but swapping the order of the first two rules for $\mathtt{rtest}_{a,b}$.
(3) Fix a stream of binary digits. Whenever more than one of the first two rules for $\mathtt{rtest}_{a,b}$ is applicable, use the next digit of the stream to decide which should be applied.
(4) Fix a stream of binary digits and a stream of natural numbers. Whenever a term of the form $\mathtt{rtest}_{a,b}(M)$ is found, read a natural number n from the second stream, then apply rule 13 for $\mathtt{rtest}_{a,b}$ n times. If only one of the two rules 11 and 12 become applicable, apply it. If both are applicable, use the next digit from the first stream to decide which of them to apply. If neither is applicable, repeat the same procedure.

It is easy to see that for any closed term M of real-number type, there is at least one term N such that $M \Rightarrow N$, and hence there is at least one term N such that $M \Rrightarrow N$. Hence, because the relation $\Rrightarrow$ is assumed to be single valued, there is a unique infinite reduction sequence $M = M_0 \Rrightarrow M_1 \Rrightarrow M_2 \Rrightarrow M_3 \Rrightarrow \cdots$. By the following lemma, if M_i is of the form $\mathtt{cons}_{a_i}(M'_i)$ then M_{i+1} must be of the form $\mathtt{cons}_{a_{i+1}} M'_{i+1}$ with $a_i \sqsubseteq a_{i+1}$. For a closed term M of ground type other than $\mathtt{I}$, such a reduction may be finite, leading to a truth value or natural number, or infinite leading to divergence.

Lemma 6.3 *If a term M is of the form $\mathtt{cons}_a M'$ and $M \Rrightarrow^* N$ then N is of the form $\mathtt{cons}_b N'$ with $a \sqsubseteq b$.*

PROOF. By case analysis of the reduction rules for $\mathtt{cons}_a$. According to the complete set of rules that define the operational semantics [11], if the reduction is in zero steps we are done, otherwise there are two cases:

(1): If $\mathtt{cons}_a(\mathtt{cons}_b N') \Rrightarrow \mathtt{cons}_{ab} N'$, then M' is of the form $\mathtt{cons}_b N'$ with $a \sqsubseteq ab$. Hence N is of the form $\mathtt{cons}_{ab} N'$,

(2): If $\mathtt{cons}_a M' \Rrightarrow \mathtt{cons}_a M''$ and $M' \Rrightarrow M''$, then N has to be of the form $\mathtt{cons}_a M''$ for $M' \Rrightarrow N'$, and hence we can take $b = a$. □

We modify the definition of operational meaning (Definition 5.2) as follows.

Definition 6.4 *For a strategy $\Rrightarrow$ and closed term M of type* $\mathtt{I}$*, we define*

$$[M]^{\Rrightarrow} = \bigsqcup\{a \in \mathtt{I} \mid \exists M'.M \Rrightarrow^* \mathtt{cons}_a M'\}.$$

If this set is non-empty, then Lemma 6.3 shows that it is an increasing chain, and hence the supremum exists. Notice that this is not a subset of $\mathtt{I}$*, as in Definition 5.2, but rather an element of* $\mathtt{I}$*.*

By a value of type `Bool` *or* `Nat` *we mean a constant for a truth value or a natural number, and values are ranged over by the letter v. For a closed term of any of these two types, we define*

$$[M]^{\Rrightarrow} = \bigsqcup\{v \mid M \Rrightarrow^* v\}.$$

The set of which the supremum is taken is either empty or a singleton because $\Rrightarrow$ is single valued.

Definition 6.5 *We define* ***strong convergence****, for closed terms, by induction on types as follows:*

(1) A closed term M of ground type is strongly convergent if for every strategy $\Rrightarrow$ as in Definition 6.2, its operational meaning $[M]^{\Rrightarrow}$ is total (i.e. a singleton interval, a truth-value, or a natural number).

(2) A closed term M of type $\sigma \to \tau$ is strongly convergent whenever MN is strongly convergent for every strongly convergent closed term N of type σ.

We henceforth refer to strong convergence simply as convergence for the sake of brevity.

The following observation is immediate.

Lemma 6.6

(1) A term $M\colon \mathtt{I}$ is convergent iff for every strategy $\Rrightarrow$ and every $\epsilon > 0$ there are an interval a of length smaller than ϵ and a term N such that $M \Rrightarrow^* \mathtt{cons}_a N$.

(2) A term M is convergent iff N is convergent whenever $M \Rrightarrow^* N$.

Lemma 6.7 *A term $\mathtt{cons}_c(M)$ is convergent iff M is convergent.*

PROOF. ($\Rightarrow$) Let $M = M_1 \Rrightarrow M_2 \Rrightarrow M_3 \Rrightarrow \cdots$ be an infinite reduction sequence and let $\epsilon > 0$. We must find n such that M_n is of the form $\texttt{cons}_d N'$ with $\kappa_d < \epsilon$. Consider the reduction

$$\texttt{cons}_c(M) = N_1 \Rrightarrow N_2 \Rrightarrow N_3 \Rrightarrow \cdots ,$$

and $\delta = \epsilon \times \kappa_c$. By hypothesis $\texttt{cons}_c(M)$ is convergent so there is i such that N_i is of the form $\texttt{cons}_b N''$ with $\kappa_b < \delta$. Hence there should be j such that M_j is of the form $\texttt{cons}_e N'''$ and $\texttt{cons}_c(M_j) \Rrightarrow \texttt{cons}_b N''$, which means that $\kappa_c \kappa_e = \kappa_b < \delta$ and hence $\kappa_e < \frac{\delta}{\kappa_c} = \frac{\epsilon \times \kappa_c}{\kappa_c} = \epsilon$.

($\Leftarrow$) Let $\texttt{cons}_c(M) = N_1 \Rrightarrow N_2 \Rrightarrow N_3 \Rrightarrow \cdots$ be an infinite reduction sequence and let $\epsilon > 0$. We must find n such that N_n is of the form $\texttt{cons}_d N'$ with $\kappa_d < \epsilon$. Consider the reduction $M = M_1 \Rrightarrow M_2 \Rrightarrow M_3 \Rrightarrow \cdots$ and $\delta = \epsilon / \kappa_a$. Because M is convergent, there is i such that M_i is of the form $\texttt{cons}_b(M')$ with $\kappa_b < \delta$. Hence, there should be j such that N_j is of the form $\texttt{cons}_e(M'')$ with $\kappa_e \leq \kappa_a \kappa_b$ and

$$\kappa_e \leq \kappa_a \kappa_b < \kappa_a \cdot \delta = \kappa_a \cdot (\epsilon / \kappa_a) = \epsilon. \qquad \square$$

To show that $\texttt{tail}_a$ is convergent, we need some lemmas. Whenever we talk about rules in the following lemmas, we assume that these rules are taken from the operational semantics.

Lemma 6.8

(1) *For all $a, b \in \mathcal{I}$, if $b \not\sqsubseteq a$ then one of the conditions in rules 3–6 holds.*

(2) *For any $a \in \mathcal{I}$ and any convergent $M\colon \texttt{I}$ there are $b \not\sqsubseteq a$ and N such that $M \Rrightarrow^* \texttt{cons}_b(N)$.*

PROOF. The first item is easily verified. For the second, let $\epsilon = \kappa_a/2$. Because M is convergent, there are b of length smaller than ϵ and N such that $M \Rrightarrow^* \texttt{cons}_b(N)$. If we had $b \sqsubseteq a$, then the length of b would be bigger than that of a, which is not the case by construction. $\square$

Lemma 6.9 *If M is convergent then,*

(1) *$\texttt{tail}_a(M) \Rrightarrow^* L$ for some convergent term L, by finitely many applications of rule 7 followed by an application of one of the rules 3–5, or*

(2) *$M \Rrightarrow^* \texttt{cons}_b(N)$ and $\texttt{tail}_a(M) \Rrightarrow^* \texttt{cons}_{(a \sqcup b)\backslash a}(\texttt{tail}_{(a \sqcup b)\backslash b}(N))$ for some convergent term N, by finitely many applications of rule 7 followed by an application of rule 6.*

PROOF. By Lemma 6.8, after finitely many applications of rule 7 to the term $\mathtt{tail}_a(M)$, we will have reductions $M \Rrightarrow^* \mathtt{cons}_b(N)$ and

$$\mathtt{tail}_a M \Rrightarrow^* \mathtt{tail}_a(\mathtt{cons}_b(N)),$$

and one of the rules 3–6 will apply to the resulting term. If one of the rules 3–5 applies then $\mathtt{tail}_a(M)$ reduces to one of the terms $\mathtt{Ycons}_L$, $\mathtt{Ycons}_R$, $\mathtt{cons}_{b\backslash a}(N)$, which are convergent, and we can let L be the corresponding term. Otherwise it reduces by rule 6 to the term $\mathtt{cons}_{(a\sqcup b)\backslash a}(\mathtt{tail}_{(a\sqcup b)\backslash b}(N))$. Because $M \Rrightarrow^* \mathtt{cons}_b N$ and M is convergent, so are $\mathtt{cons}_b N$ and N. □

Lemma 6.10 *The term* $\mathtt{tail}_a$ *is convergent.*

PROOF. Let M be convergent, consider the reduction

$$\mathtt{tail}_a(M) = N_0 \Rrightarrow N_1 \Rrightarrow N_2 \Rrightarrow \cdots ,$$

and let r_i be the label of the rule that justifies the reduction $N_i \Rrightarrow N_{i+1}$. By Lemma 6.9, if there is i such that r_i is one of 3–5, then $\mathtt{tail}_a(M)$ is convergent, and otherwise the sequence $(r_i)_i$ belongs to the set of words $7^*6(7^*61)^\omega$. We have to argue that in the second case $\mathtt{tail}_a(M)$ is also convergent. Let n_i be the sequence such that the sequence r_i can be written as $7^{n_0}\,6\,\prod_i(7^{n_{i+1}}61)$.

By hypothesis, the term $M_0 = M$ is convergent, and if M_i is convergent then

$$M_i \Rrightarrow^* \mathtt{cons}_{c_i}(M_{i+1})$$

for a unique interval c_i and a unique term M_{i+1} by finitely many applications of rule 2, and M_{i+1} must also be convergent. This inductively defines sequences c_i and M_i, and it is easy to see that, for any i,

$$M \Rrightarrow^* \mathtt{cons}_{c_0 c_1 \ldots c_i}(M_{i+1}).$$

Now, using the sequence c_i, inductively define

$$\beta_0 = (a \sqcup c_0) \setminus c_0, \qquad \alpha_0 = (a \sqcup c_0) \setminus a,$$
$$\beta_{i+1} = (\beta_i \sqcup c_{i+1}) \setminus c_{i+1}, \qquad \alpha_{i+1} = (\beta_i \sqcup c_{i+1}) \setminus \beta_i.$$

A routine argument by induction on i shows that

$$\mathtt{tail}_a(M) \Rrightarrow^* \mathtt{cons}_{\alpha_0\alpha_1\cdots\alpha_i}(\mathtt{tail}_{\beta_i}(M_{i+1})),$$

as illustrated below:

$$\begin{aligned}
\mathtt{tail}_a(M) = N_0 \overset{7}{\Rrightarrow}{}^* M_{n_0} &= \mathtt{tail}_a(\mathtt{cons}_{c_0}(M_1)) \\
\overset{6}{\Rrightarrow} N_{n_0+1} &= \mathtt{cons}_{\alpha_0}(\mathtt{tail}_{\beta_0}(M_1)) \\
\overset{7}{\Rrightarrow}{}^* N_{n_1} &= \mathtt{cons}_{\alpha_0}(\mathtt{tail}_{\beta_0}\mathtt{cons}_{c_1}(M_2)) \\
\overset{6}{\Rrightarrow} N_{n_1+1} &= \mathtt{cons}_{\alpha_0}\mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1}(M_2)) \\
\overset{1}{\Rrightarrow} N_{n_1+2} &= \mathtt{cons}_{\alpha_0\alpha_1}(\mathtt{tail}_{\beta_1}(M_2)) \\
&\vdots \\
\overset{1}{\Rrightarrow} N_{n_i+2} &= \mathtt{cons}_{\alpha_0\alpha_1\cdots\alpha_i}(\mathtt{tail}_{\beta_i}(M_{i+1})).
\end{aligned}$$

Now let $\epsilon > 0$, and define $\epsilon' = \kappa_a/\epsilon$. Because M is convergent, there is i such that $\kappa_{c_0c_1\ldots c_i} < \epsilon'$ and hence $\kappa_a/\kappa_{c_0c_1\ldots c_i} < \epsilon$. An easy proof by induction on i shows that $\kappa_a/\kappa_{c_0c_1\ldots c_i} = \kappa_{\alpha_0\alpha_1\cdots\alpha_i}$, which shows that $\mathtt{tail}_a(M)$ is convergent. □

As application, we show how the program `Average`, defined in Section 2 can be proved to be correct using the denotational semantics and the notion of strong convergence. More examples, including multiplication, division, and absolute value, among others, are developed in the first-named author's PhD thesis [22] using the same techniques.

Lemma 6.11 *The term* $\mathtt{rtest}_{b,c}$ *is convergent.*

PROOF. Let $N \colon \mathtt{I}$ be a convergent term. Consider $\epsilon = (c - b)/2$. Because N is convergent, there are an interval a of length smaller than ϵ and a term M such that $N \Rrightarrow^* \mathtt{cons_a} M$. For such an interval, at least one of the conditions needed to apply the rules (11) or (12) holds, and hence $\mathtt{rtest}_{b,c}(N) \Rrightarrow^+ v$ for some truth value v.

6.1 Total Correctness of the Average Program

In view of computational adequacy, partial correctness of the program can be formulated as follows:

Lemma 6.12 $[\![\mathtt{Average}]\!](\eta(x), \eta(y)) = \eta(x \oplus y)$ *for all total* $x, y \in \mathcal{I}$.

To prove this, we use the following lemma. As usual, a recursive program is interpreted as the least fixed point of a functional extracted from the program. For the program `Average`, we denote this functional by $\Phi : D \to D$ where, according to the denotational interpretation of types, D has to be the domain $(\mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I})$. Then $[\![\mathtt{Average}]\!] = \bigsqcup_n \mathsf{Average}_n$, where $\mathsf{Average}_n = \Phi^n(\bot)$.

Lemma 6.13 *For all total $x, y \in \mathcal{I}$, the following conditions hold:*

(1) $[\![\mathsf{Average}_n]\!](\eta(x), \eta(y))$ *is of the form* $\downarrow F_n$ *for* $F_n \subseteq \mathcal{I}$ *finite,*
(2) $\kappa_z \leq \left(\frac{4}{3}\right)^{-n+1}$ *for each* $z \in F_n$*,*
(3) $F_n \sqsubseteq \eta(x \oplus y)$*.*

PROOF. The proof is by induction on n.

1. $n = 0$. We know that $\mathsf{Average}_0(\eta(x), \eta(y)) = \{\bot\} = \downarrow\{\bot\}$ for any $x, y \in [0, 1]$. Take $z \in F_n = \{\bot\}$, so $\kappa_z = 1 < (4/3)^{-n+1} = (4/3)$, and $\{\bot\} \sqsubseteq_H \eta(x \oplus y)$ for all $x, y \in [0, 1]$.

2. Assume that it holds for n. To show that it holds for $n+1$, we proceed according to the position of x and y relative to the points $l = 1/4$ and $r = 3/4$ used in the definition of the average program. All cases are handled in a similar way. We consider the case $x \leq 1/4$ and $y \leq 1/4$ as a representative example.

$$\begin{aligned}\mathsf{Average}_{n+1}(\eta(x), \eta(y)) &= \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y))))\\ &= \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))),\end{aligned}$$

and by the induction hypothesis, $\mathsf{Average}_n(\eta(t), \eta(s))$ is of the form $\downarrow F_n$ for F_n finite, $t = \mathsf{tail}_L(x)$ and $s = \mathsf{tail}_L(y)$. Take $F_{n+1} = \widehat{\mathsf{cons}}_L(F_n)$. Then

$$\mathsf{Average}_{n+1}(\eta(x), \eta(y))$$

is of the form $\downarrow\widehat{\mathsf{cons}}_L(F_n)$. Because F_n is finite, so is F_{n+1}.

To show that $\kappa_z \leq (\frac{4}{3})^{-n}$ for any $z \in F_{n+1}$, let $t \in F_n$ such that $z = \mathsf{cons}_L(t)$. By the induction hypothesis $\kappa_t \leq (\frac{4}{3})^{-n+1}$. We have $z = \mathsf{cons}_L(t) = \frac{3t}{4}$, and hence

$$\overline{t} - \underline{t} \leq \left(\frac{4}{3}\right)^{-n+1}$$

$$\frac{3}{4}\overline{t} - \frac{3}{4}\underline{t} \leq \left(\frac{3}{4}\right)\left(\frac{4}{3}\right)^{-n+1} = \left(\frac{4}{3}\right)^{-n}$$

and so $\kappa_z \leq (\frac{4}{3})^{-n}$.

To show that $F_{n+1} \subseteq \eta(x \oplus y)$, again let $z \in F_{n+1}$ and $t \in F_n$ such that such that $z = \mathsf{cons}_L(t)$. By the induction hypothesis $t \in \eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_L(y))$, hence

$$\begin{aligned}z = \mathsf{cons}_L(t) &\in \widehat{\mathsf{cons}}_L(\eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_L(y)))\\ &= \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{4x}{3} \oplus \frac{4y}{3}\right)\right) = \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{4x+4y}{6}\right)\right)\\ &= \eta\left(\mathsf{cons}_L\left(\frac{4x+4y}{6}\right)\right) = \eta\left(\left(\frac{3}{4}\right)\left(\frac{4x+4y}{6}\right)\right)\\ &= \eta\left(\frac{x+y}{2}\right) = \eta(x \oplus y).\end{aligned}$$

as required. □

To conclude, we establish convergence of $\mathtt{Average}$.

Lemma 6.14 *For any two convergent terms $N_1, N_2 : \mathtt{I}$, there are an interval a of length $3/4$ and two convergent terms N_1', N_2' such that $\mathtt{Average}(N_1, N_2) \Rrightarrow^+ \mathtt{cons}_a(\mathtt{Average}(N_1', N_2'))$.*

PROOF. To reduce $\mathtt{Average}(N_1, N_2)$, we must first unfold the definition, and then reduce $\mathtt{rtest}_{1/4,3/4}(N_1)$, repeatedly applying rule 10, until we get a truth value, which is possible by Lemma 6.11 because N_1 has been assumed to be convergent. At this point, we have to apply one of the rules 8 or 9. In either case, we will next have to reduce $\mathtt{rtest}_{1/4,3/4}(N_2)$ until it becomes a truth value. Then again one of the two rules 8 and 9 will have to be applied, which clearly leads to a term of the form $\mathtt{cons}_a\mathsf{Average}(\mathtt{tail}_{b_1}N_1, \mathtt{tail}_{b_2}N_2)$ with $\kappa_a = 3/4$. By Lemma 6.10, we can take $N_1' = \mathtt{tail}_{b_1}N_1$ and $N_2' = \mathtt{tail}_{b_2}N_2$. □

Lemma 6.15 *The term $\mathtt{Average}$ is convergent.*

PROOF. Let N_1 and N_2 be convergent terms of type $\mathtt{I}$. By repeatedly applying Lemma 6.14 and rules 1 and 2, we conclude that for every n there are an interval a of length $(3/4)^n$ and a term M such that $\mathtt{Average}(N_1, N_2) \Rrightarrow^+ \mathtt{cons}_a(M)$. Here we use the fact that the length of the interval concatenation bc is the product of the lengths of the intervals b and c in connection with rule 1. □

Lemma 6.12 amounts to commutativity of the diagram

$$\begin{tikzcd}
I \times I \arrow[r, hook] \arrow[d, "\oplus"'] & \mathcal{I} \times \mathcal{I} \arrow[r, hook] & \mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \arrow[d, "{[\![\mathtt{Average}]\!]}"] \\
I \arrow[r, hook] & \mathcal{I} \arrow[r, hook] & \mathcal{P}^H\mathcal{I},
\end{tikzcd}$$

where $I = [0, 1]$ and the horizontal arrows are the obvious inclusions. The results of Escardó, Hofmann and Streicher [9] show that the diagram cannot be completed with a sequentially computable down arrow $\mathcal{I} \times \mathcal{I} \to \mathcal{I}$. Thus, we overcome the problem by allowing our program to be multi-valued at partial inputs. Lemma 6.13 shows that the single-valued output of the program at a total input arises as the least upper bound of multi-valued partial outputs. In other words, there are different computation paths that give different, but consistent partial results at finite stages, but all of them converge to the same total real number.

Several other examples of recursive definitions, including multiplication and division, are developed in [22], with total correctness proofs following the above pattern.

7 Conclusion and Further Work

Our running example illustrates two important ideas discussed in the introduction:

(1) By considering a multi-valued or non-deterministic construction, it is possible to have sequential programs for important functions that only admit parallel realizations in the (singled-valued) interval-domain model, overcoming the problem identified by Escardó, Hofmann and Streicher [9].
(2) In order to obtain total correctness from partial correctness, a generalization of the notion of termination is needed in the case of real-number computations.

Regarding 1, we conjecture that all computable first-order functions are definable in the language. We have some partial results regarding definability of second-order computable functionals such as definite integration. This will be reported elsewhere, but we remark that the ideas regarding 2 are applied for that purpose.

It is an open problem to find a denotational semantics that would allow to prove total correctness without the need of resorting to operational methods such as strong convergence. As we have seen, the Plotkin and Smyth powerdomains cannot be used for that purpose either. In fact, the results of Section 4 immediately imply that even other powerdomains such as the sandwich and the mixed powerdomain cannot be used. Moreover, it is easy to verify that any of the known powerdomains which do not arise as the composition of powerdomains with the Hoare powerdomain as the last component in the composition are ruled out.

Acknowledgements. We thank Achim Jung, Paul Levy, Steve Vickers and Andrew Moshier for comments and suggestions.

References

[1] Samson Abramsky and Achim Jung, Domain Theory, in: S. Abramsky and D. Gabbay and T. S. E. Maibaum, eds., *Handbook of Logic in Computer Science Volume 3* (Oxford University Press, 1994) 1–168.

[2] E. Bishop, and D. Bridges, *Constructive Analysis* (Springer, Berlin, 1985).

[3] H. J. Boehm and R. Cartwright, Exact Real Arithmetic: Formulating Real Numbres as functions, in: Turner. D., editor, *Research Topics in Functional Programming* (Addison-Wesley 1990) 43–64.

[4] V. Brattka, Recursive characterization of computable real-valued functions and relations, *Theoretical Computer Science* **162** (1996) 45–77.

[5] Peter Buneman and Susan Davidson and Aaron Watters, A semantics for complex objects and approximate queries, *JCSS* **43** (1991) 170–218.

[6] Abbas Edalat and Peter John Potts and Philipp Sünderhauf, Lazy Computation with Exact Real Numbers, *International Conference on Functional Programming* (1998) 185–194.

[7] M. H. Escardó, Real PCF extended with ∃ is universal, in: A. Edalat and S. Jourdan and G. McCusker, eds., *Advances in Theory and Formal Methods of Computing: Proceedings of the Third Imperial College Workshop* (Christ Church, Oxford, 1996) 13–24.

[8] M. H. Escardó PCF extended with real numbers: A domain-theoretic approach to higher-order exact real number computation, *PhD thesis at Imperial College of the University of London* 1997.

[9] M. H. Escardó and M. Hofmann and Th. Streicher, On the non-sequential nature of the interval-domain model of exact real-number computation, *Mathematical Structures in Computer Science* Accepted for publication (2002).

[10] M. H. Escardó and Th. Streicher, Induction and recursion on the partial real line with applications to Real PCF, *Theoretical Computer Science* **210 (1)** (1999) 121–157.

[11] M. H. Escardó, PCF Extended with Real Numbers, *Theoretical Computer Science* **162 (1)** (1996) 79–115.

[12] A. Farjudian, Sequentiality and Piece-wise affinity in Segments of Real-PCF, *Electronic Notes in Theoretical Computer Science* **73** (2004) 3–4

[13] A. Farjudian, Sequentiality in Real Number Computation, *PhD thesis at the University of Birmingham* 2004.

[14] Pietro Di Gianantonio, A Functional Approach to Computability on Real Numbers *PhD thesis* (Udine, 1993).

[15] Pietro Di Gianantonio, An Abstract Data Type for real numbers, *Theoretical Computer Science* **221** (1999) 295-326

[16] G. Gierz and et al., *Continuous lattices and domains*, (Cambridge University Press, 2003).

[17] C. A. Gunter, The Mixed Powerdomain, *Theoretical Computer Science* **103 (2)** (1992) 311–334.

[18] C. A. Gunter and D. S. Scott, Semantic Domains, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science* **B** (1990) 633–674.

[19] Reinhold Heckmann, Power Domain Constructions, *Science of Computer Programming* **17 (1-3)** (1991) 77–117

[20] H. Luckhardt, A fundamental effect in computations on real numbers, *Theoretical Computer Science* **5 (3)** (1977/78) 321–324.

[21] E. Manes Monads of Sets in: M. Hazewinkel, editor, *Handbook of Algebra* **3** (Elsevier Science, 2003) 67–153.

[22] José R. Marcial-Romero, Semantics of a sequential language for exact real-number computation, *PhD thesis* (Birmingham, December, 2004).

[23] N. Th. Müller, The iRRAM: Exact Arithmetic in C++, in: Blanck, Jens and Brattka, Vasco and Hertling, Peter, *Computability and Complexity in Analysis* **2064** (LNCS, 2001) 222–252.

[24] D. Normann, Exact real number computations relative to hereditarily total functionals, *Theoretical Computer Science* **284 (2)** (2002) 437–453.

[25] G. D. Plotkin, A Powerdomain Construction, *SIAM Journal on Computing* **5 (3)** (1976) 452–487.

[26] G. D. Plotkin, LCF Considered as a Programming Language, *Theoretical Computer Science* **5 (1)** (1977) 223–255.

[27] G. D. Plotkin, Domains *Post-graduate Lecture in Advanced Domain Theory Univesity of Edinburgh, Departament of Computer Science. Available from the author's web page* (1983), pages 116.

[28] Peter John Potts and Abbas Edalat and Martín Hötzel Escardó, Semantics of Exact real arithmetic, *Proceedings 12th IEEE Symposium on Logic in Computer Science* (1997) 248–257.

[29] Peter John Potts, Exact real arithmetic using Möbius Transformations, *PhD thesis at Imperial College of the University of London* 1998.

[30] Dana Scott, Lattice theory, data type and semantics, in: Randall Rustin, editor, eds., *Formal Semantics of Algorithmic Languages* (Prentice Hall, 1972) 65–106.

[31] M. B. Smyth, Power Domains, *Journal of Computer and System Science* **16** (1978) 23–36.

[32] M. B. Smyth, Powerdomains and predicate transformers: A topological view *ICALP '83, LNCS* **154** (Springer, 1983) 662–675.

[33] M. B. Smyth, Topology, in: S. Abramsky, D. M. Gabbay, and T.S.E Maibaum, eds., *Handbook on Logic in Computer Science* **1** (1992) 641–761.

[34] S. Vickers, *Topolgy via Logic* (Cambridge University Press, Cambridge, 1989).

[35] K. Weihrauch, *Computable Analysis* (Springer-Verlag, 2000) .