Semantics of a Sequential Language for Exact Real-Number Computation

J. Raymundo Marcial-Romero   Martín H. Escardó
University of Birmingham, Birmingham B15 2TT, England
{jrm,mhe}@cs.bham.ac.uk

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.*

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 to say that the computability issues are well understood [17]. 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 that articulate the data-abstraction requirement. Based on ideas arising from constructive mathematics, Boehm and Cartwright [3] proposed a compelling operational solution to the problem. However, the 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 of solving this problem has been developed by Edalat, Potts and Sünderhauf [5], as discussed below.

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 this research programme, we pause to discuss previous work.

Di Gianantonio [11], Escardó [6], and Potts et al. [24] have introduced various extensions of the programming language PCF with a ground type for real numbers. In the three cases, the real numbers type is interpreted as a variation of the interval domain introduced by Scott [25]. In the presence of a certain parallel conditional [22], all computable first-order functions on the reals are definable in the languages [11, 7]. By further adding Plotkin's parallel existential quantifier [22], all computable functions of all orders become definable in the languages [11, 7, 9]. 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 [20].

It is natural to ask whether the presence of such parallel constructs is an artifact of the languages or they are needed for intrinsic reasons. Escardó, Hofmann and Streicher [8] have shown that, in interval 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 [10] 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 the following. Because computable functions on the reals are continuous (see e.g. [17]), and because the real line is a connected space, any computable boolean-valued relation on the reals is constantly true or constantly false unless it diverges for some inputs. Hence definitions using the sequential conditional produce constant total functions or else partial functions. If one allows the boolean-valued relations to diverge at some inputs, then non-trivial predicates are obtained, and this together with the parallel conditional allow us to define non-trivial total functions [6].

This phenomenon had been anticipated by Boehm and Cartwright [3], who also proposed a solution to this 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 $a < x$ and $x < b$ 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. [18]), and is thus determined by the operational semantics.

As an 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 postponed to the body of the paper. One of our main results is that the Hoare powerdomain gives a computationally adequate denotational semantics. Unfortunately, the Plotkin and Smyth powerdomains do not even make the rtest construction continuous. 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. 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). The technique is illustrated by a proof of total correctness of our sequential program for addition. Further applications are discussed in the concluding section.

Related work. Edalat, Potts and Sünderhauf [5] have 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 e.g. 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 that 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 deterministic or not. For our domain of partial real numbers, we consider the interval domain proposed by Scott [25], but the work reported here is expected to apply to many possible notions of domain of partial real numbers.

Di Gianantonio [12] 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.

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 a sample application. Section 7 summarizes the main results and discusses open problems and ongoing work.

2. Running example

In order to motivate 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 work 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 [6], the average operation $(- \oplus -)\colon [0,1] \times [0,1] \to [0,1]$ defined by

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

can be implemented as follows:

$$
\begin{array}{l}
x \oplus y = \texttt{pif}\ \ x < c \\
\qquad\quad \texttt{then pif}\ \ y < c \\
\qquad\qquad\quad \texttt{then cons}_L(\texttt{tail}_L(x) \oplus \texttt{tail}_L(y)) \\
\qquad\qquad\quad \texttt{else cons}_C(\texttt{tail}_L(x) \oplus \texttt{tail}_R(y)) \\
\qquad\quad \texttt{else pif}\ \ y < c \\
\qquad\qquad\quad \texttt{then cons}_C(\texttt{tail}_R(x) \oplus \texttt{tail}_L(y)) \\
\qquad\qquad\quad \texttt{else cons}_R(\texttt{tail}_R(x) \oplus \texttt{tail}_R(y))\texttt{.}
\end{array}
$$

Here

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

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

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

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

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

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

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

The intuition behind this 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

$$
\begin{array}{l}
1/2 \oplus 1/2 = \texttt{pif}\ \perp \\
\qquad\quad \texttt{then pif}\ \perp \\
\qquad\qquad\quad \texttt{then cons}_L(1\ \oplus\ 1) \\
\qquad\qquad\quad \texttt{else cons}_C(1\ \oplus\ 0) \\
\qquad\quad \texttt{else pif}\ \perp \\
\qquad\qquad\quad \texttt{then cons}_C(0\ \oplus\ 1) \\
\qquad\qquad\quad \texttt{else cons}_R(0\ \oplus\ 0)\texttt{.}
\end{array}
$$

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 is exponential 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:

$$
\begin{array}{l}
\texttt{Average}(x, y) = \\
\texttt{if rtest}_{l,r}(x) \\
\quad \texttt{then} \\
\qquad \texttt{if rtest}_{l,r}(y) \\
\qquad\quad \texttt{then cons}_L(\texttt{Average}(\texttt{tail}_L(x), \texttt{tail}_L(y))) \\
\qquad\quad \texttt{else cons}_C(\texttt{Average}(\texttt{tail}_L(x), \texttt{tail}_R(y))) \\
\quad \texttt{else} \\
\qquad \texttt{if rtest}_{l,r}(y) \\
\qquad\quad \texttt{then cons}_C(\texttt{Average}(\texttt{tail}_R(x), \texttt{tail}_L(y))) \\
\qquad\quad \texttt{else cons}_R(\texttt{Average}(\texttt{tail}_R(x), \texttt{tail}_R(y)))\texttt{,}
\end{array}
$$

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 $\texttt{rtest}$, 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.

3. Background

For domain-theoretic concepts, the reader is referred to [1, 23], and for topological concepts to [28, 29] (see also [13]). Here we briefly summarize the notions and facts that are relevant for 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

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

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 said 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* 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 $\perp \sqsubseteq \mathsf{false}, \perp \sqsubseteq \mathsf{true}$.

3.2. The Interval Domain

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

$$x \sqsubseteq y \text{ iff } x \supseteq y,$$

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

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

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

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

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

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

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 [21, 26, 27] are usually constructed as ideal completions [15] of finite subsets of basis elements. For our purposes, it is more convenient to work with their topological representations [23, 1, 16], 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 [13]. 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

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

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

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

which is also a topological embedding. Instead of considering multiplication, one can equivalently consider the extension operator [19, Proposition 2.14], in this case given by

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

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:

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

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 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. □

Thus, only powerdomains satisfying the above two inequalities qualify. 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 [14] and the sandwich [4] powerdomains also fail 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 2. *There is a continuous extension* $\mathrm{rtest}^H_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ *of the function* $\mathrm{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$.

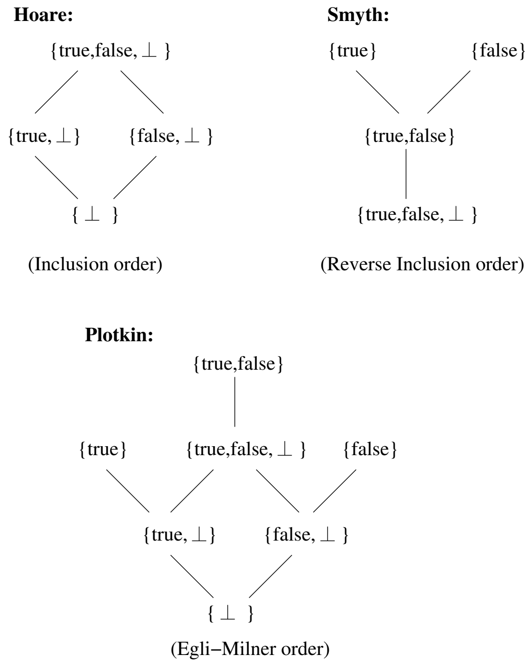

Figure 1. Powerdomains of $\mathbb{T}_\perp$.

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

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

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

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. $\square$

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 $\mathrm{rtest}^H_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ 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 $\mathsf{rtest}_{a,b}\colon \mathcal{I} \to \mathcal{P}\mathbb{T}_\perp$.

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 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 (and indeed constructive) counterpart of this function would be. Hence we stick to the construction as originally proposed by Boehm and Cartwright.

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 [6] 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.

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

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

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.

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

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

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

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

Operational Semantics We consider a small-step style operational semantics for our language. The rules are those given in [6], with the cases for the parallel conditional removed and the following added:

1. $\texttt{rtest}_{b,c}(\texttt{cons}_a\ M) \to \texttt{true}$ if $\overline{a} < c$,
2. $\texttt{rtest}_{b,c}(\texttt{cons}_a\ M) \to \texttt{false}$ if $b < \underline{a}$,
3. $\texttt{rtest}_{b,c}\ M \to \texttt{rtest}_{b,c}\ M'$ if $M \to M'$.

Notice that if the interval a is contained in the interval $[b, c]$ the first two rules can be applied. Of course, the role of the third rule is to obtain more information when the first and second rules cannot be applied. For that purpose, the following rule from [6] plays a crucial role:

4. $\texttt{cons}_a(\texttt{cons}_b M) \to \texttt{cons}_{ab} M$.

Here the interval ab is defined to be $\mathsf{cons}_a(b)$, where cons is the extension to the interval domain of the function defined in Section 2. This rule is justified by the associativity law $a(bx) = (ab)x$. 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.

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

Definition 3. Firstly, we define the operational meaning of closed terms M of ground type γ in i steps of computation, written $[M]_i$, which is to be an element of the domain $[\![\gamma]\!]$.

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

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

if this set is non-empty, and $[M]_i = \perp$ otherwise. Here the relation $\xrightarrow{k}$ denotes the k-fold composition of the relation $\to$.

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

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

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

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 4. $[\![M]\!] = \uplus \{[\![N]\!] \mid M \to N\}$.

Proof. By structural induction on M. □

Lemma 5. *For every j,* $[\![M]\!] = \uplus \{[\![N]\!] \mid M \xrightarrow{j} N\}$.

Proof. By induction on the length j of the evaluation using the previous lemma. □

Lemma 6 (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 $M \overset{i}{\to} \mathtt{cons}_a M'$. Because $\widehat{\mathsf{cons}_a}[\![M']\!] = [\![\mathtt{cons}_a M']\!]$, Lemma 5 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 $x \in [\![M']\!]$. □

In order to establish completeness, we proceed as in [22, 6].

Definition 7. *We define a notion of computability for closed terms by induction on types as follows:*

- *A closed term M of ground type is computable whenever $[\![M]\!] \sqsubseteq [M]$,*
- *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 8. *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. □

Lemma 9. *For any continuous function $f : D \to E$ of continuous dcpos, if $y \ll f(x)$ then there is $x' \ll x$ with $y \ll f(x')$.*

Proof. See [1] or [13]. □

Lemma 10 (Completeness). *Every term is computable.*

Proof. The proof is by structural induction on the formation rules of terms. Here we only consider the constant $\mathtt{rtest}_{a,b}$. We have to show that if M is computable, then so is $\mathtt{rtest}_{a,b}M$.

Assume that $[\![\mathtt{rtest}_{a,b}M]\!] \neq \bot$ for computable M of type $\mathtt{I}$. Let $P \ll [\![\mathtt{rtest}_{a,b}M]\!]$. We need to show that there is i with $P \sqsubseteq [\mathsf{rtest}_{a,b}M]_i$. Because $\mathtt{rtest}_{a,b}[\![M]\!] = \overline{\mathsf{rtest}}_{a,b}[\![M]\!]$, Lemma 9 shows that there is $X \ll [\![M]\!]$ with $P \ll \overline{\mathsf{rtest}}_{a,b}X$. As M is computable, there is j such that $X \sqsubseteq [M]_j$.

We proceed by cases on P:

(a) $P = \bot$: immediate.

(b) $P = \{\mathsf{true}, \bot\}$: Then $\{\mathsf{true}, \bot\} \sqsubseteq \overline{\mathsf{rtest}}_{a,b}X$ and hence $\{\mathsf{true}, \bot\} \sqsubseteq \overline{\mathsf{rtest}}_{a,b}[M]_j$ because $\overline{\mathsf{rtest}}_{a,b}$ is monotone and $X \sqsubseteq [M]_j$. This means that $\mathsf{true} \in \overline{\mathsf{rtest}}_{a,b}[M]_j$, and hence there is $t \in [M]_j$ such that $\mathsf{true} \in \mathsf{rtest}_{a,b}(t)$. By definition of $[M]_j$, $t \in [M]_j$ implies that there is M' with $M \overset{k}{\to} \mathtt{cons}_t M'$ for $k \leq j$. Because $\mathsf{true} \in \mathsf{rtest}_{a,b}(t)$, we conclude that $t < b$, so we have a reduction $\mathtt{rtest}_{a,b}(\mathtt{cons}_t M') \overset{1}{\to} \mathtt{true}$ and then a reduction $\mathtt{rtest}_{a,b}M \overset{k}{\to} \mathtt{rtest}_{a,b}(\mathtt{cons}_t M') \overset{1}{\to} \mathtt{true}$. Hence $\mathtt{rtest}_{a,b}M \overset{k+1}{\to} \mathtt{true}, k \leq j$. and we can take $i = k + 1$.

(c) $P = \{\mathsf{false}, \bot\}$. Similar to (b).

(d) $P = \{\mathsf{true}, \mathsf{false}, \bot\}$: Then $\{\mathsf{true}, \mathsf{false}, \bot\} \sqsubseteq \overline{\mathsf{rtest}}_{a,b}X$ and $\{\mathsf{true}, \mathsf{false}, \bot\} \sqsubseteq \overline{\mathsf{rtest}}_{a,b}[M]_j$ because $\overline{\mathsf{rtest}}_{a,b}$ is monotone and $X \sqsubseteq [M]_j$. This means that $\mathsf{true} \in \overline{\mathsf{rtest}}_{a,b}[M]_j$ and $\mathsf{false} \in \overline{\mathsf{rtest}}_{a,b}[M]_j$, and hence there are $t, s \in [M]_j$ such that $\mathsf{true} \in \mathsf{rtest}_{a,b}(t)$ and $\mathsf{false} \in \mathsf{rtest}_{a,b}(s)$. By definition of $[M]_j$, we conclude that there are M', M'' and $l, l' \leq j$ such that $M \overset{l}{\to} \mathtt{cons}_t M'$ and $M \overset{l'}{\to} \mathtt{cons}_s M''$. Because $\mathsf{true} \in \mathsf{rtest}_{a,b}(t)$, we deduce that $t < b$. We then have a reduction $\mathtt{rtest}_{a,b}(\mathtt{cons}_t M') \overset{1}{\to} \mathtt{true}$, from which it follows that $\mathtt{rtest}_{a,b}M \overset{l}{\to} \mathtt{rtest}_{a,b}(\mathtt{cons}_t M') \overset{1}{\to} \mathtt{true}$, and hence that $\mathtt{rtest}_{a,b}M \overset{l+1}{\to} \mathtt{true}$. Because $\mathsf{false} \in \mathsf{rtest}_{a,b}(s)$, we conclude that $a < s$, and so we have a reduction $\mathtt{rtest}_{a,b}(\mathtt{cons}_s M'') \overset{1}{\to} \mathtt{false}$, from which it follows that $\mathtt{rtest}_{a,b}M \overset{l'}{\to} \mathtt{rtest}_{a,b}(\mathtt{cons}_s M'') \overset{1}{\to} \mathtt{false}$, and so $\mathtt{rtest}_{a,b}M \overset{l+1}{\to} \mathtt{true}$ and $\mathtt{rtest}_{a,b}M \overset{l'+1}{\to} \mathtt{false}$. Hence we can take $i = l + l' + 2$. □

Soundness and completeness show that

Theorem 11. *Computational adequacy holds.*

6. Program Correctness

In this section, we assume that during evaluation, no redex can be infinitely delayed. Intuitively, this means that the reduction rule 3 for $\mathtt{rtest}_{a,b}$ cannot be applied infinitely often, giving rise to a divergent computation, unless rules 1 and 2 are never applicable. This restriction to the operational semantics could have been imposed in the formulation of computational adequacy, but it does not make any difference, because $\perp$ is always a possible denotational value of any term.

Computational adequacy allows us to prove partial correctness, but given that denotationally $\perp$ is always a possible result, it does not allow us to prove total correctness. To overcome this problem, we introduce a generalization of the notion of termination.

Definition 12. *A term M of type* `I` *is strongly convergent if for every infinite reduction sequence*

$$M \to M_1 \to M_2 \to \dots,$$

and every $\epsilon > 0$ there is i such that M_i is of the form $\mathtt{cons}_b M'$ with $|b| < \epsilon$.

We henceforth refer to strong convergence as simply convergence for the sake of brevity. The following is immediate from definition 12.

Lemma 13. *A term M of type* `I` *is convergent iff for every reduction sequence*

$$M_0 \to^+ M_1 \to^+ M_2 \to^+ M_3 \to^+ \cdots,$$

with $M = M_0$ and every $\epsilon > 0$ there is i such that M_i is of the form $\mathtt{cons}_a M'$ with $|a| < \epsilon$.

Lemma 14. *If a term M is of the form $\mathtt{cons}_a M'$ and $M \to^* N$ 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 [6], if the reduction is in zero steps we are done, otherwise there are two cases:

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

The following lemma is immediate:

Lemma 15. *If a term M is convergent and $M \to^* N$ then N is convergent.*

Lemma 16. *If M is a convergent term then $\mathtt{cons}_a(M)$ and $\mathtt{tail}_a(M)$ are convergent.*

Proof. The proof is by induction over the number of reduction steps. The first is trivial, but the second is not. ☐

As an application, we indicate how the program `Average` defined in Section 2 can be shown to be correct using our denotational semantics and the notion of strong convergence. In view of computational adequacy, partial correctness of the program can be formulated as follows:

Lemma 17. $[\![\mathtt{Average}]\!](\eta(x), \eta(y)) = \eta(x \oplus y)$ *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 $\mathsf{Average}_n = \Phi^n(\perp)$.

Lemma 18. *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. $|z| \leq \left(\frac{4}{3}\right)^{-n+1}$ *for each $z \in F_n$,*
3. $F_n \sqsubseteq \eta(x \oplus y)$.

To conclude, we need to establish the following:

Lemma 19. *If N_1, N_2 :* `I` *are convergent terms then* $\mathtt{Average}(N_1, N_2)$ *is convergent.*

Lemma 17 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 [8] 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 18 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.

7. Conclusion and Ongoing Work

The sample application given in Section 6 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 [8].

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 know that, in fact, all computable first-order functions are definable in the language, and 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 which 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 cannot be used.

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

References

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science Volume 3*, pages 1–168. Oxford University Press, 1994.

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

[3] H. Boehm and R. Cartwright. Exact real arithmetic: Formulating real numbres as functions. *In Turner. D., editor, Research Topics in Functional Programming*, pages 43–64, 1990.

[4] P. Buneman, S. Davidson, and A. Watters. A semantics for complex objects and approximate queries. pages 305–314, 1988.

[5] A. Edalat, P. J. Potts, and P. Sünderhauf. Lazy computation with exact real numbers. In *International Conference on Functional Programming*, pages 185–194, 1998.

[6] M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, 1996.

[7] M. H. Escardó. Real PCF extended with $\exists$ is universal. In A. Edalat, S. Jourdan, and G. McCusker, editors, *Advances in Theory and Formal Methods of Computing: Proceedings of the Third Imperial College Workshop, April 1996*, pages 13–24, Christ Church, Oxford, 1996. IC Press.

[8] M. H. Escardó, M. Hofmann, and T. 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).

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

[10] A. Farjudian. Sequentiality and piece-wise affinity in segments of real-pcf. ENTCS (to appear).

[11] P. D. Gianantonio. *A Functional Approach to Computability on Real Numbers*. PhD thesis, Udine, 1993.

[12] P. D. Gianantonio. An abstract data type for real numbers. *Theoretical Computer Science*, 221(1–2):295–326, 1999.

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

[14] C. A. Gunter. The mixed powerdomain. *Theoretical Computer Science*, 103(2):311–334, 1992.

[15] C. A. Gunter and D. S. Scott. Semantic domains. *Handbook of Theoretical Computer Science*, B:633–674, 1990. edited by J. van Leeuwen, North Holland.

[16] R. Heckmann. Power domain constructions. *Science of Computer Programming*, 17(1-3):77–117, 1991.

[17] K.Weihrauch. *Computable Analysis*. Springer-Verlag, 2000.

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

[19] E. Manes. Monads of sets. In M. Hazewinkel, editor, *Handbook of Algebra*, volume 3, pages 67–153. Elsevier Science, 2003.

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

[21] G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[22] G. D. Plotkin. Lcf considered as a programming language. *Theoretical Computer Science*, 5(1):223–255, 1977.

[23] 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*, page 116, 1983.

[24] P. J. Potts, A. Edalat, and M. H. Escardó. Semantics of exact real arithmetic. In *Proceedings 12^{th} IEEE Symposium on Logic in Computer Science*, pages 248–257, 1997.

[25] D. Scott. Lattice theory, data type and semantics. In *In Randall Rustin, editor, Formal Semantics of Algorithmic Languages,*, pages 65–106. Prentice Hall, 1972.

[26] M. B. Smyth. Power domains. *Journal of Computer and System Science*, 16:23–36, 1978.

[27] M. B. Smyth. Powerdomains and predicate transformers: A topological view. In *ICALP '83, LNCS*, volume 154, pages 662–675. Springer, 1983.

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

[29] S. Vickers. *Topology Via Logic*. Cambridge University Press, Cambridge, 1989.