School of Computer Science
The University of Birmingham

Semantics of a Sequential Language for Exact Real-Number Computation

José Raymundo Marcial-Romero

Dr. Martín Hötzel Escardó
Supervisor

Prof. M. Andrew Moshier
External Examiner

Prof. Achim Jung
Internal Examiner

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

December 2004

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 several examples.

A Gabriela

"Por todo el apoyo brindado durante estos tres años"

Acknowledgements

While studying for my fisrt degree at the Universidad Autónoma de Puebla, my supervisor, José de Jesús Lavalle Martínez persuaded me to consider the field of theoretical computer science. Lavalle, together with Jesús García-Fernandez, Guillermo de Ita-Luna, Arturo Díaz-Pérez, José Juan Palacios-Perez, Alma Lizbeth Juarez-Dominguez and César Bautista-Ramos, encouraged me to continue studying for a postgraduate degree. I am grateful to all of them. I am especially grateful to Lavalle for having advised me in difficult situations.

I am also extremely grateful to my supervisor Martín Hötzel Escardó, for all his help and encouragement. He introduced me to domain theory, topology and computable analysis. He spent so many hours explaining each of these subjects in detail that he deserves most of the credit for my work.

I benefited from many discussions about domain theory with Achim Jung, who gave me several suggestions to improve this work. I had also interesting discussions with Paul Levy when I was learning his call-by-push value paradigm.

Sponsorship from the Mexican Institute *Consejo Nacional de Ciencia y Tecnología* (CONACYT) made this work possible.

The complementary support given by the Mexican organisation Secretaría de Educación Pública (SEP) made this work possible also.

During the workshop Domain VI, Thomas Streicher and Andrej Bauer gave me valuable feedback. While visiting Martin Escardó, Reinhold Heckmann explained in full the small and big powerdomains from which the last part of Chapter 7 was obtained. I am grateful to all of them.

During these three years at the University of Birmingham, I have had valuable discussions with Amin Farjudian. I am grateful to Felipe Orihuela-Espina, Weng K. Ho, the English for International Students Unit and my supervisor for proof-reading some of the chapters of this thesis. Both Jung and Moshier have carefully proof-read this thesis. Moshier suggested a simplification of the proof of completeness in Chapter 9, which is included in this thesis.

Last but not least, I would like to thank my parents for their unconditional support.

Contents

List of Figures

Chapter 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 [8]. It is fair to say that the computability issues are well understood [86]. 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 [10], 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 Edalat, Potts and Sünderhauf [21], as discussed below.

In the light of the above, we present a programming language which fulfils the following requirements simultaneously:

1. The language should be expressive for practical purposes, and ideally as expressive as the theory permits.

2. The data type of real numbers should be abstract.

3. The language should have a sequential evaluation strategy.

Before elaborating this research programme, we pause to discuss previous work.

Di Gianantonio [15], Escardó [22], and Potts et al. [72] have introduced various extensions of the programming language PCF with a ground type for

real numbers. Each of these researchers interprets the real numbers type as a variation of the interval domain introduced by Scott [74]. In the presence of a certain parallel conditional [68], all computable first-order functions on the reals are definable in the languages [15, 23]. By further adding Plotkin's parallel existential quantifier [68], all computable functions of all orders become definable in the languages [15, 23, 28]. 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 have been developed by Normann [64].

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 [27] have shown that, in the 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 [29] 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. [86]), 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 either constant total functions or 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 the non-trivial total functions [22].

This phenomenon had been anticipated by Boehm and Cartwright [10], who also proposed a solution to the problem. In this thesis, 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. [56]), and is thus determined by the operational semantics.

As applications of the construction, we give several examples of recursive definition of *sequential* programs, which are 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 thesis. 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 thesis.

As is well known, Hoare semantics can be used in order to establish *partial* correctness only [81]. 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 giving proofs of total correctness of some sequential programs. Further applications are discussed in the concluding section. The main results of this work have been published in [57].

1.1 Brief summary of contributions

We can summarize the results presented in this thesis as follows:

- *Main contribution of the thesis.* The development of an expressive programming language with an abstract data type of real numbers and a sequential operational semantics.

- *Technical contribution of the thesis.* The presentation of a denotational semantics for the multi-valued construction proposed by Boehm and Cartwright [10]. Some positive and negative results are derived, as follows:

  Positive Results

  - The proof that the Hoare powerdomain can be used to model the multi-valued construction.
  - Computational adequacy between the operational and denotational semantics.
  - The proof of the total correctness of the programs by decomposing it in partial correctness, which is obtained by denotational arguments, and strong convergence, which is achieved using operational arguments. This decomposition is needed due to the fact that the Hoare powerdomain only allows the partial correctness of programs to be proved.
  - Several applications of the technique.

  Negative Results

  - We prove that the Smyth and Plotkin powerdomain cannot be used to model the construction. Moreover, other well known powerdomains such as the Sandwich and Mixed powerdomains cannot be used.

1.2 Related work.

Edalat, Potts and Sünderhauf [21] 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, 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 present study, 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 [74], but the present findings are expected to apply to many possible notions of domain of partial real numbers.

Di Gianantonio [18] 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 introduce effectivity for functions into the theory of continuous computability, Brattka [11] introduces a class of real-valued recursive relations. What makes his approach relational, as in our case, rather than functional is a multi-valued operator defined among the basic recursive definitions. The main difference between his and our approach is that he introduces a notion of continuity of relations and constructs recursive continuous relations, while we work with continuous functions in a powerdomain setting and construct recursive continuous functions. An implementation of Brattka's approach has been presented by Müller [62] in the programming language C++. Unlike our approach, Müller's does not consider a domain theoretical model for his language.

1.3 Organization

This thesis is divided into four parts.

In Part I, the background needed to understand the main body of the thesis is presented. This part is divided into four chapters.

In Chapter 2, we present the domain theory's material needed in the thesis. Domain theory is used to give a denotational semantics of the programming language presented in Chapter 9.

In Chapter 3, we introduce a domain which can be used to study real numbers. In this model, a real number is considered as the limit of a computable sequence of intervals of the real line. In this sense, intervals are seen as partial realizations of real numbers.

In Chapter 4, we present a summary of the different approaches to real number computation. This chapter is included especially for readers without knowledge of the different alternatives to real number computation. The last section of this chapter describes exact real-number computation, which is the approach considered in this thesis.

In Chapter 5, we introduce the programming languages PCF and Real PCF. PCF is the base for the programming language presented in Chapter 9, where PCF is extended with a data-type for real numbers. Real PCF is an extension of PCF with real numbers. We introduce Real PCF in this work to discuss the similarities and differences with the language presented in Chapter 9.

In Part II, we present a model for sequential computation on the real numbers. This part is divided into three chapters.

In Chapter 6, we present a detailed description of the problem to be solved, and discuss our approach to its solution. The last section of the chapter introduces the main construction studied in the thesis, called $\mathsf{rtest}_{a,b}$.

In Chapter 7, we present the mathematical interpretation of the construction $\mathsf{rtest}_{a,b}$. We describe the powerdomain constructions available in the literature which fail and finally use one which can be considered as the mathematical interpretation of the construct. In the last section we consider possible alternatives to be used in the powerdomain setting.

In Chapter 8, we introduce recursive definitions using the $\mathsf{rtest}_{a,b}$ constructor. In this chapter we present three additional constructors in order to define recursive operations. We present definitions for addition, multiplication and division, among others.

In Part III, we present a programming language for sequential computation over the real numbers. Three chapter are included in this part.

In Chapter 9, the programming language for non-deterministic test is presented. The language's constructs are those of PCF, plus two constructs of Real PCF, plus the $\mathsf{rtest}_{a,b}$ construct introduced in Chapter 6. We present a denotational semantics based on a powerdomain construction and prove computational adequacy of the operational semantics with respect to the denotational semantics.

In Chapter 10, we introduce a notion of strong convergence of programs. We present the equivalent programs of the definitions described in Chapter 8 and prove that they converge to the desire solution.

Finally, in Part IV, we present the conclusions of the thesis. We summarize the results obtained in Chapter 11 and discuss further problems and work still to be done in Chapter 12.

Part I

Background

Chapter 2

Domain Theory

In this chapter, we present the results from domain theory needed to understand the thesis. The main references of this chapter are Abramsky and Jung [2], and Plotkin [66]. We use domain theory as a mathematical tool to give denotational semantics of the programming language presented in Chapter 9. We begin with some of the most basic definitions.

2.1 Basic definitions

Definition 2.1.1 (preorder). *A set P together with a binary relation $\sqsubseteq$ on P is called a **preorder** denoted by $(P, \sqsubseteq_P)$ if the following holds for all $x, y, z \in P$:*

1. $x \sqsubseteq x$ *(reflexivity)*
2. $x \sqsubseteq y \ \wedge \ y \sqsubseteq z \Rightarrow x \sqsubseteq z$ *(transitivity)*

Definition 2.1.2 (poset). *A **partially ordered set** $(P, \sqsubseteq_P)$ (poset) is a preorder which satisfies an extra condition for all $x, y \in P$:*

3. $x \sqsubseteq y \ \wedge \ y \sqsubseteq x \Rightarrow x = y$ *(antisymmetry)*

Examples of posets are given in Figure 2.1. When no confusion arises, we simply write P for the preorder or poset $(P, \sqsubseteq_P)$. Sometimes, the principal interest resides in the set of elements above or below an element or a subset of a set.

Definition 2.1.3 (upper set). *Let P be a poset, a subset A of P is an **upper set** if $x \in A$ implies $y \in A$ for all $y \sqsupseteq x$.*

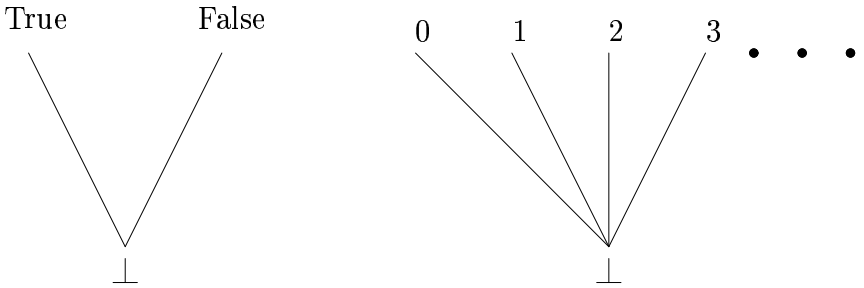


Figure 2.1: Posets

The set of all elements above some element of a set A is denoted by $\uparrow A$. $\uparrow\{x\}$ is written as $\uparrow x$. The dual notions are **lower set**, $\downarrow A$ and $\downarrow\{x\} = \downarrow x$.

Similar to upper or lower set, the interest may reside on the set of elements in between two points; such sets are called convex.

Definition 2.1.4 (convex set). *Let P be a poset, a subset A of P is **convex** iff, whenever $a, c \in A$ and $a \sqsubseteq b \sqsubseteq c$ holds, then $b \in A$.*

Definition 2.1.5 (upper bound). *Let P be a poset, an element $x \in P$ is called an **upper bound** for a subset A of P, if x is above every element of A.*

A **lower bound** is defined dually. If for a subset A of P the set of upper bounds has a least element x, this x is called the **least upper bound** (lub) or supremum; the **greatest lower bound** (glb) or infimum is defined dually for the set of lower bounds.

Definition 2.1.6 (directed set). *A subset D of a poset P is called **directed** provided*

1. *D is non-empty,*
2. *$x, y \in D \Rightarrow \exists z \in D$ such that $x \sqsubseteq z \ \wedge \ y \sqsubseteq z$.*

We write $\bigsqcup^{\uparrow} A$ to denote the lub of a directed set A.

Definition 2.1.7 (lattice). *A **lattice** is a poset P in which*

- *there is a least element (commonly denoted by $\bot$) and a greatest element (commonly denoted by $\top$);*
- *each pair $x, y \in P$ has a lub $x \vee y$ and a glb $x \wedge y$.*

This is equivalent to say that, each finite subset of P has both a lub and a glb.

Definition 2.1.8 (complete lattice). *A **complete lattice** is a lattice in which every subset has both a lub and a glb.*

Each half of the condition is sufficient for Definition 2.1.8: if in a poset P every subset has a lub, then it is also true that every subset has a glb, and vice versa.

A weaker notion of "completeness", Definition 2.1.9, is also of great interest:

Definition 2.1.9 (dcpo). *A **directed-complete poset**(dcpo) is a poset P in which every directed subset $D \subseteq P$ has a lub.*

Definition 2.1.10 (cpo). *A **complete poset**(cpo) is a dcpo P having a least element (sometimes it is called a dcpo with a bottom element). The bottom element is typically represented by $\bot_P$.*

When the cpo P is understood from the context, the subscript on $\bot_P$ will usually be dropped. The examples of Figure 2.1 are easily seen to be cpos.

2.2 Maps between dcpos

In this thesis, the kind of maps of interest between dcpos and cpos are those which are continuous. In what follows, the results are mainly stated in terms of dcpo, but they can be applied to cpos as well.

Definition 2.2.1 (continuous map). *Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be dcpos. A map $f : (P, \sqsubseteq_P) \to (Q, \sqsubseteq_Q)$ is (order-)**continuous** if*

- $\forall x, y \in P : x \sqsubseteq_P y \Rightarrow f(x) \sqsubseteq_Q f(y)$ *(f is monotone),*
- *for all directed sets $D \subseteq (P, \sqsubseteq_P)$, $\bigsqcup^{\uparrow}(f[D]) = f(\bigsqcup^{\uparrow} D)$.*

*Continuous maps are sometimes called, in the literature, **morphisms** or **continuous functions**.*

Definition 2.2.2 (strict maps). *A map $f : (P, \sqsubseteq_P) \to (Q, \sqsubseteq_Q)$ between cpos is said to be **strict** if $f(\bot_P) = \bot_Q$.*

For two dcpos P and Q, let $[P \to Q]$ be the set of all continuous maps from P to Q. $[P \to Q]$ is ordered pointwise, i.e. $f \sqsubseteq g$ iff $f(x) \sqsubseteq g(x)$ holds for all $x \in X$.

An important fact about the set of all continuous maps between dcpos can be stated as:

Proposition 2.2.3. *If P and Q are dcpos, then $[P \to Q]$ is a dcpo under the pointwise order.*

Proof. We show that the least upper bound $\bigsqcup_{f \in D} f$ of a directed set $D \subseteq [P \to Q]$ is the map $g : x \mapsto \bigsqcup^{\uparrow}_{f \in D} f$. This make sense as a map because the directedness of D means $\{f(x) \mid f \in D\}$ is directed for each x. It is easy to verify that it is monotone. To see that it is continuous, suppose $R \subseteq P$ is directed. Then

$$g(\bigsqcup_{x \in R} x) = \bigsqcup_{f \in D} f(\bigsqcup_{x \in R} x) = \bigsqcup_{x \in R} \bigsqcup_{f \in D} f(x) = \bigsqcup_{f \in D} \bigsqcup_{x \in R} f(x) = \bigsqcup_{x \in R} f(x).$$

Now, clearly $f(x) \sqsubseteq g(x)$ for each $f \in D$. If $h \sqsupseteq f$ for each $f \in D$, then $h(x) \sqsupseteq \bigsqcup_{f \in D} f(x)$ for each $x \in P$ so $h \sqsupseteq g$. □

2.3 Least fixed points

There is a key theorem on which much of the discussion of the semantics of programming languages is based. The fixed point theorem asserts that every continuous self-map on cpos has a least fixed point.

Theorem 2.3.1 (Least fixed point). *If P is a cpo and $f : P \to P$ is continuous, then it has a least fixed point $\mathsf{fix}(f) \in P$. That is,*

- $\mathsf{fix}(f) = f(\mathsf{fix}(f))$, *and*
- $\mathsf{fix}(f) \sqsubseteq x$ *for any $x \in P$ such that $x = f(x)$.*

A more general result from which Theorem 2.3.1 follows as a corollary is given below. We require a definition:

Definition 2.3.2. *Let P be a poset. An ω-**chain** $(x_n)_{n \in \omega}$ in P is a set of elements $x_n \in P$ such that $x_n \sqsubseteq x_m$ whenever $n \leq m$. A poset P is ω-**complete** (and hence an ω-**cpo**) if every ω-chain has a least upper bound. Given ω-cpo's P and Q, a monotone map $f : P \to Q$ is said to be ω-**continuous** if $f(\bigsqcup_{n \in \omega} x_n) = \bigsqcup_{n \in \omega} f(x_n)$ for any ω-chain $(x_n)_{n \in \omega}$.*

Theorem 2.3.3 (Fixed points in ω-cpo's). *Suppose P is an ω-cpo and $f : P \to P$ is ω-continuous. If $x \sqsubseteq f(x)$ for some $x \in P$, then there is a least element $y \in P$ such that*

1. *$y = f(y)$, and*
2. *$y \sqsubseteq z$ for any $z \in P$ such that $z = f(z)$ and $x \sqsubseteq z$.*

Proof. By an induction on n using the monotonicity of f, it is easy to see that $f^n(x) \sqsubseteq f^{n+1}(x)$ for every n so $(f^n(x))_{n\in\omega}$ is an ω-chain. Set $y = \bigsqcup_{n\in\omega} f^n(x)$. To see property (1), calculate

$$f(y) = f(\bigsqcup_{n\in\omega} f^n(x)) = \bigsqcup_{n\in\omega} f^{n+1}(x) = \bigsqcup_{n\in\omega} f^n(x) = y.$$

To verify (2), suppose z is a fixed point of f and $x \sqsubseteq z$. Then, for each $n \in \omega$, $f^n(x) \sqsubseteq f^n(z) = z$, so $y = \bigsqcup_{n\in\omega} f^n(x) \sqsubseteq z$. □

To see that Theorem 2.3.1 is a corollary of Theorem 2.3.3, note that an ω-chain is a directed set, so a cpo is an ω-cpo and a continuous function is also ω-continuous. Since $\bot \sqsubseteq f(\bot)$, Theorem 2.3.1 follows immediately.

2.4 Algebraic and continuous dcpos

An important class of dcpos is the so-called algebraic dcpos (called in the literature algebraic domains).

Definition 2.4.1 (compact points). *An element a in a dcpo P is **compact** (or isolated, or finite) iff for all directed sets $D \subseteq P$, the relation $a \sqsubseteq \bigsqcup^{\uparrow} D$ implies $a \in {\downarrow}D$, i.e. there is an element $d \in D$ such that $a \sqsubseteq d$.*

Definition 2.4.2 (algebraic dcpo). *A dcpo P is **algebraic** iff for all $x \in P$, there is a directed set of compact points with lub x.*

The class of all algebraic dcpos is denoted by **ALG**. The set of compact points of an algebraic dcpo is called its **base**. Algebraic dcpos are characterised by a base from which all dcpo points can be generated by directed lubs.

To motivate this presentation, some examples of algebraic dcpos are given.

1. Every finite poset P is an algebraic dcpo whose base is P itself.

2. If X is any set, then the powerset of X (the set of all subsets of X) forms an algebraic dcpo when ordered by inclusion '$\subseteq$'. The compact points are just the finite sets.

The class of algebraic dcpos is generalised to the class of continuous dcpos by generalising the concept of compact points to the way-below relation. Computationally speaking, a point x is way-below a point y iff every computation converging to a point above y must eventually produce partial results above x. Formally it is defined as:

Definition 2.4.3 (way-below relation). *The* ***way-below relation*** *of a dcpo P is the binary relation $\ll$ on P defined as: $x \ll y$ in P iff for every directed set $D \subseteq P$,*

$$y \sqsubseteq \bigsqcup^{\uparrow} D \text{ implies } x \sqsubseteq a \text{ for some } a \in D.$$

The way-below relation gives enough information to define a continuous dcpo.

Definition 2.4.4 (continuous dcpo). *A dcpo P is* ***continuous*** *if it satisfies the following, for all $x \in P$:*

$$x = \bigsqcup_{y \ll x}^{\uparrow} y.$$

That is, the set of elements way-below x is directed and has x as its least upper bound. The category of continuous dcpos and continuous maps is denoted by **CONT**.

There is a very important feature of the way-below relation, the **interpolation property**.

Theorem 2.4.5 (interpolation property). *Let X be a continuous dcpo. For every two points x and z in X with $x \ll z$, there is a point $y \in X$ such that $x \ll y \ll z$.*

For a point $x \in X$, we write $\mathop{\downarrow}\!\!\!\!\!\downarrow x$ to denote the set of elements way-below x, i.e. the set $\{y \mid y \ll x\}$ and $\uparrow\!\!\!\!\!\uparrow x = \{y \mid x \ll y\}$ dually.

An example of a dcpo that is continuous but not algebraic is the following: Let $X = [0, 1]$ be the unit interval of the real line ordered by the usual order such that 0 is the least element and 1 is the greatest element. X forms a complete lattice since every subset has a supremum. In X, $x \ll y$ holds iff $x < y$ or $x = y = 0$. X is continuous since $x = \bigsqcup\{y \mid y < x\}$ holds for all $x > 0$. It is however not algebraic because it contains only one compact point, namely 0, and this single point is not capable to generate all points by directed lubs.

Lemma 2.4.6. *For any continuous function $f : D \to E$ of continuous dcpos, if $y \ll f(x)$ then $\exists x' \ll x$ such that $y \ll f(x')$.*

Proof. Let f continuous and $y \ll f(x)$, by the interpolation property, $\exists y' \in E$ such that $y \ll y' \ll f(x)$. As E is continuous, $f(x) = \bigsqcup\{f(x') \mid x' \ll x\}$ and this set is directed, so there is some $f(x')$ such that $y' \sqsubseteq f(x')$. Then $y \ll y'$ implies $y \ll f(x')$. □

Definition 2.4.7 (domain). *A **domain** is a continuous cpo.*

Definition 2.4.8 (basis). *A **basis** of a continuous cpo P is a subset B such that $\forall x \in P$,*

$$x = \bigsqcup^{\uparrow} \!\!\downarrow y \cap B.$$

That is, x is the directed join of the members of B which are way-below x.

Lemma 2.4.9. *If D is a domain, C a finitely generated closed subset of D and $f : D \to D$ Scott continuous, then*

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

Proof. That $\downarrow\{f(x) \mid x \in C\} \subseteq \mathsf{cl}\{f(x) \mid x \in C\}$ is given by definition of cl. For $\supseteq$, we must show that $\downarrow\{f(x) \mid x \in C\}$ is directed closed. Let $T \subseteq \downarrow\{f(x) \mid x \in C\}$ be a directed set. For every $t \in T$, we have that $t \sqsubseteq f(x_t)$, where $x_t \in C$. Then by continuity of f, $\bigsqcup t \sqsubseteq \bigsqcup f(x_t) = f(\bigsqcup x_t)$. As C is a finitely generated closed set then $\bigsqcup x_t \in C$, hence $\bigsqcup t \in \downarrow\{f(x) \mid x \in C\}$. □

Some other useful definitions are the following:

- A continuous cpo is **countably based** if it has a countable basis.
- A **continuous lattice** is a continuous dcpo which is also a complete lattice.
- A subset of a poset is **bounded** if it has an upper bound.
- A poset is a **bounded complete poset** if every bounded subset has a lub.
- A **bounded complete continuous cpo** is a continuous cpo which is also bounded complete.
- A bounded complete, countably based continuous cpo is referred as a **continuous Scott Domain**.
- The class of dcpos together with continuous maps between them form a cartesian closed category denoted by **DCPO**.

2.5 Powerdomains

In this section we describe a collection of operators in domains which are useful for the denotational semantics of non-deterministic constructors [41, 42, 40, 76, 67, 1, 4, 43]. Actually, we introduce five such operators. We will define an analog to the powerset operation on sets. In the domain theory literature they are known as powerdomains. The source of this section is [2].

Powerdomains can be charaterised algebraically, so we begin with some basic concepts from universal algebra.

Definition 2.5.1. *A **signature** $\Sigma = \langle \Omega, \alpha \rangle$ consists of a set Ω of operation symbols and a map $\alpha : \Omega \to \mathbb{N}$, assigning to each operation symbol a (finite) arity.*

Definition 2.5.2. *A Σ-algebra $\underline{A} = \langle A, I \rangle$ is given by a carrier set A and an interpretation I of the operation symbols, in the sense that for $f \in \Omega$, $I(f)$ is a map from $A^{\alpha(f)}$ to A. We also write f_A for the interpreted operation symbol.*

Definition 2.5.3. *A homomorphism between two Σ-algebras $\underline{A}$ and $\underline{B}$ is a map $h : A \to B$ which commutes with the operations:*

$$\forall f \in \Omega, \quad h(f_A(a_1, \ldots, a_{\alpha(f)})) = f_B(h(a_1), \ldots, h(a_{\alpha(f)})).$$

We denote the term algebra over X with respect to a signature Σ by $T_\Sigma(X)$. It has the universal property that each map from X to A, where $\underline{A} = \langle A, I \rangle$ is a Σ-algebra, can be extended uniquely to a homomorphism $\bar{h} : T_\Sigma(X) \to \underline{A}$. Let V be a fixed countable set whose elements we refer to as 'variables'. Pairs of elements of $T_\Sigma(V)$ are used to encode equations. An equation $\tau_1 = \tau_2$ is said to hold in an algebra $\underline{A} = \langle A, I \rangle$ if for each map $h : V \to A$ we have $\bar{h}(\tau_1) = \bar{h}(\tau_2)$. The pair $\langle \bar{h}(\tau_1), \bar{h}(\tau_2) \rangle$ is also called an instance of the equation $\tau_1 = \tau_2$.

Definition 2.5.4. *A **dcpo-algebra** is characterized by the property that the carrier set is equipped with an order relation such that it becomes a dcpo, and such that each operation is Scott-continuous.*

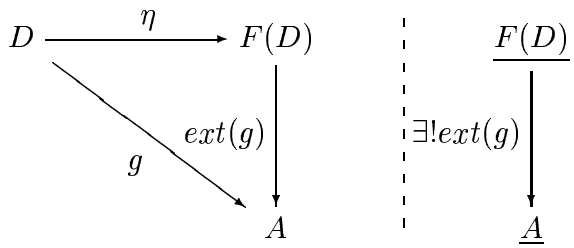
Because of the order we also can incorporate inequalities. So we let the pair $\langle \tau_1, \tau_2 \rangle \in \varepsilon \subseteq T_\Sigma(V) \times T_\Sigma(V)$ stand for the inequality $\tau_1 \sqsubseteq \tau_2$.

Notation 2.5.5. *By $\boldsymbol{DCPO}(\Sigma, \varepsilon)$ we mean the category of Scott-continuous homomorphisms and dcpo-algebras over the signature Σ which satisfy the inequalities of ε.*

Definition 2.5.6. *A **(continuous) domain-algebra** is a dcpo-algebra whose carrier set forms a continuous domain.*

Notation 2.5.7. *By $\boldsymbol{CONT}(\Sigma, \varepsilon)$ we mean the category of Scott continuous homomorphisms and domain-algebras over the signature Σ which satisfies the inequalities of ε.*

Theorem 2.5.8. *Suppose a signature Σ and a set ε of inequalities has been fixed, then given a continuous domain D there exists a dcpo-algebra $\underline{F(D)}$, whose carrier set $F(D)$ is a continuous domain which satisfies the inequalities in ε, and a Scott-continuous function $\eta : D \to F(D)$ such that given any such domain-algebra $\underline{A}$ and Scott-continuous map $g : D \to A$ there is a unique Scott continuous homomorphism $ext(g) : \underline{F(D)} \to \underline{A}$ for which $ext(g) \circ \eta = g$ (as illustrated in the following diagram)*

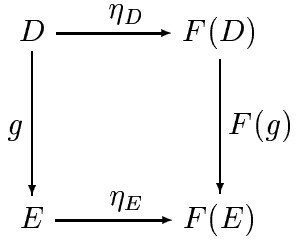


Proof. See [2]. □

Corollary 2.5.9. *For any signature Σ and set ε of inequalities the forgetful functor $U : \mathbf{CONT}(\Sigma, \varepsilon) \to \mathbf{CONT}$ has a left adjoint F.*

Proof. See [2]. □

The action of the left adjoint functor on morphisms is obtained by assigning to a continuous function $g : D \to E$ the homomorphism which extends $\eta_E \circ g$.



Notation 2.5.10. *In our applications we write*

$$\overline{g} \text{ instead of } ext(g),$$

$$\widehat{g} \text{ instead of } F(g).$$

There is a detailed account of powerdomains in the literature [67, 78, 79, 2, 35, 36, 73]. In this thesis, the use of powerdomains, with the exception of the Hoare powerdomain, is limited to finite cpos, in particular the flat domain of booleans. For this reason and avoiding further explanations, we write:

- $\mathcal{P}^S D$ for the Smyth powerdomain of a domain D, also called the upper powerdomain.
- $\mathcal{P}^P D$ for the Plotkin powerdomain of an ω-continuous dcpo D, also called the convex powerdomain, and
- $\mathcal{P}^H D$ for the Hoare powerdomain of a domain D, also called the lower powerdomain.

There are at least two other powerdomain constructions considered in the literature written as:

- ΣD for the Sandwich powerdomain of an algebraic dcpo D, and
- σD for the Mixed powerdomain of an algebraic dcpo D.

Powerdomains are usually constructed as ideal completions of finite subsets of basis elements [34]. For our purposes, it is more convenient to work with both their algebraic representations [2] and their topological representations [66, 2, 35]. We need some auxiliary definitions from topology in order to fulfil our purposes.

2.5.1 Topology

Definition 2.5.11 (Topological space). *A topological space* $\mathbf{X}$ *is a non-empty set* X *together with a set* Γ *of subsets of* X *such that*

- $\emptyset$ *and* X *are in* Γ*,*
- *Arbitrary unions of members of* Γ *are again in* Γ*,*
- *Finite intersections of members of* Γ *are again in* Γ*.*

As with posets, we often identify both the topological space $\mathbf{X}$ and its topology Γ simply by X.

The sets in Γ are called **open sets**, their complements are called **closed sets**.

Given a topological space X, we denote by Δ the set of closed sets in X. Δ is needed to define topological closure.

Definition 2.5.12. *The **topological closure** of a subset A of a topological space X is defined as $\bigcap\{B \in \Delta \mid B \supseteq A\}$.*

Definition 2.5.13. *The **interior** of a subset H of a topological space X, denoted by H°, is the union of all subsets of H which are open in X.*

The definition of a continuous map between topological spaces can be stated as follows:

Definition 2.5.14. *Given topological spaces X, Y, a map $f : X \to Y$ is continuous if the inverse image of every open set in Y is open in X.*

There is a method in which cpos can be seen as topological spaces.

Definition 2.5.15 (Scott open set). *Let P be a cpo. A subset A of P is called **Scott open** if*

- *it is an upper set, and*
- *for each directed set $D \subseteq P$, $\bigsqcup^{\uparrow} D \in A$ implies the existence of an element $a \in D \cap A$.*

The Scott open subsets of P form a topology which is called the **Scott topology** on P.

It is a well known fact in domain theory that order continuity coincides with Scott continuity. Hence, in order to prove continuity of maps between Scott domains, we can use either the order theoretical definition or the topological definition.

We define the Scott closed sets in a dcpo P as the complements of the Scott open sets. For clarity, we give the definition explicitly.

Definition 2.5.16 (Scott closed set). *A subset A of a dcpo P 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.*

Notation 2.5.17. *We use $\mathsf{cl}(A)$ to denote the **topological closure** of A, i.e. the smallest Scott closed set containing A.*

The closure of a singleton set $\mathsf{cl}\{x\}$ will often be abbreviated by $\mathsf{cl}(x)$.

Definition 2.5.18. *A **cover** of a set A is a collection $\mathcal{U}$ of sets such that $A \subset \bigcup_{U \in \mathcal{U}} U$. A subcover of a given cover $\mathcal{U}$ for A is a subcollection $\mathcal{V} \subset \mathcal{U}$ which still forms a cover of A.*

Definition 2.5.19 (Scott compact set). *A set is called **Scott compact** if every cover consisting of Scott open sets has a finite subcover.*

Given a domain D, let ΩD denote its Scott topology, ΩD is said to be **coherent** if any finite intersection of compact open sets is again a compact (open) set.

Definition 2.5.20 (lense). *A **lens** is a non-empty set that arises as the intersection of a Scott closed set and a Scott compact upper set.*

On the order-convex sets of a dcpo D ($conv(D)$) we have the following definition,

Definition 2.5.21 (Egli-Milner ordering). *On $conv(D)$, the **Egli-Milner ordering**, $\sqsubseteq_{\mathsf{EM}}$ is defined by*

$$K \sqsubseteq_{\mathsf{EM}} L \text{ if } L \subseteq \uparrow K \text{ and } K \subseteq \mathsf{cl}(L).$$

2.5.2 Plotkin powerdomain

Definition 2.5.22. *The **Plotkin powertheory** is defined by a signature with one binary operation $\uplus$ and the equations*

- $A \uplus B = B \uplus A$ *(commutativity),*
- $A \uplus (B \uplus C) = (A \uplus B) \uplus C$ *(associativity),*
- $A \uplus A = A$ *(idempotence).*

The operation $\uplus$ is called formal union.
A dcpo-algebra with respect to this theory is called a **dcpo-semilattice**.

Definition 2.5.23 (algebraic definition). *The free dcpo-semilattice over a dcpo D is called the Plotkin powerdomain $\mathcal{P}^P(D)$.*

Theorem 2.5.24. *The Plotkin powerdomain $\mathcal{P}^P D$ of a finite domain D consists of the set $conv(D)$ under the Egli-Milner order where:*

- *The formal-union operation $A \uplus B$ is given by actual union $A \cup B$ followed by topological convex closure.*
- *The natural topological embedding $\eta\colon D \to \mathcal{P}^P D$ is given by $x \mapsto \{x\}$.*
- *the extension operator is given by:*

$$\begin{array}{rccc} \overline{g}\colon & \mathcal{P}^P D & \to & A \\ & X & \mapsto & \bigsqcup_{x \in X} f(x) \end{array}$$

- *The left adjoint functor with action on continuous maps is given by:*

$$\begin{array}{rccc} \widehat{g}\colon & \mathcal{P}^P D & \to & \mathcal{P}^P E \\ & X & \mapsto & conv(f[x]) \end{array}$$

2.5.3 Smyth powerdomain

Definition 2.5.25. *If the Plotkin powertheory is augmented by the inequality*

$$A \sqsupseteq A \uplus B$$

then we obtain the ***Smyth powertheory****.*

Algebras for this theory are called deflationary semilattices.

Definition 2.5.26 (algebraic definition). *The free deflationary semilattice over a dcpo D is called the Smyth powerdomain $\mathcal{P}^S(D)$.*

Theorem 2.5.27. *The Smyth powerdomain $\mathcal{P}^S D$ of a finite dcpo D consists of the set of non-empty upper subsets ordered by* reverse *inclusion, where:*

- *The formal-union operation $A \uplus B$ is given by actual union $A \cup B$.*
- *The natural topological embedding $\eta\colon D \to \mathcal{P}^S D$ is given by $x \mapsto \uparrow x$.*
- *the extension operator is given by:*

$$\begin{array}{rccc} \overline{g}\colon & \mathcal{P}^S D & \to & A \\ & X & \mapsto & \bigUplus_{x \in X} (f[x]) \end{array}$$

- *The left adjoint functor with action on continuous maps is given by:*

$$\begin{array}{rccc} \widehat{g}\colon & \mathcal{P}^S D & \to & \mathcal{P}^S E \\ & X & \mapsto & \uparrow f[X] \end{array}$$

2.5.4 Hoare powerdomain

Definition 2.5.28. *If the Plotkin powertheory is augmented by the inequality*

$$A \sqsubseteq A \uplus B$$

then we obtain the ***Hoare powertheory****.*

Algebras for this theory are called inflationary semilattices. We drop the finiteness restriction for the Hoare powerdomain.

Definition 2.5.29 (algebraic definition). *The free inflationary semilattice over a dcpo D is called the Hoare powerdomain $\mathcal{P}^H(D)$.*

Theorem 2.5.30 (topological characterisation). *The* ***Hoare powerdomain*** *$\mathcal{P}^H D$ consists of all non-empty Scott closed subsets of D ordered by inclusion.*

1. *Least upper bounds are given by*

$$\bigsqcup_{i\in I} A_i = \mathsf{cl}\left(\bigcup_{i\in I} A_i\right).$$

2. *Formal unions are given by actual unions:*

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

3. *The natural topological embedding $\eta : D \to \mathcal{P}^H D$ is given by*

$$x \mapsto {\downarrow}x.$$

4. *the extension operator is given by:*

$$\begin{array}{rccc} \overline{g}\colon & \mathcal{P}^H D & \to & A \\ & X & \mapsto & \mathsf{cl}\left(\{\uplus\, g[f] \mid F \subseteq_{fin} X\}\right) \end{array}$$

5. *The left adjoint functor with action on continuous maps is given by:*

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

2.5.5 Sandwich powerdomain

Starting from problems in database theory, Buneman et al. [13] proposed to combine the Hoare and Smyth powerdomain to a so-called Sandwich powerdomain. Heckmann [36] also investigated this construction considering the empty set as a member of the powerdomain. His construction is called the Big powerdomain. Heckmann showed that there is an isomorphism between the Sandwich powerdomain with the empty set added and the Big powerdomain.

Definition 2.5.31. *The **Sandwich powerdomain** ΣD over an algebraic cpo D with coherent topology has carrier:*

$$\begin{aligned} \{(A^H, A^S) \mid & (1)\, A^H \text{ is a non-empty Scott closed set of } D \\ & (2)\, A^S \text{ is a non-empty Scott compact upper set of } D \\ & (3)\, A^H \subseteq \mathsf{cl}(A^S)\}. \end{aligned}$$

The elements are ordered by

$$(A^H, A^S) \sqsubseteq (B^H, B^S) \text{ iff } A^H \subseteq B^H \text{ and } A^S \supseteq B^S,$$

i.e. the order is inherited from $\mathcal{P}^H D \times \mathcal{P}^S D$.

The derived operations are given by:

- $P \uplus Q = (P^H \uplus Q^H, P^S \uplus Q^S) = (P^H \cup Q^H, P^S \cup Q^S)$,
- $\eta(x) = (\eta_H(x), \eta_S(x)) = (\downarrow(x), \uparrow(x))$,
- $extf(A) = (ext_H f^H A^H, ext_S f^S A^S) = (\bigcup_{a \in A^H}(f(a))^H, \bigcup_{a \in A^S}(f(a))^S)$.

The equation for the derived operation *map* may be computed from the definition of *ext*

- $map = ext(\eta(f))$.

2.5.6 Mixed powerdomain

The logic of the classical powerdomains was investigated by Gunter. By extending the logic of Plotkin's domain in a natural way, he developed a so-called mixed powerdomain and the theory of it [33, 32]. The original treatment of the mixed powerdomain was considered without the empty set in the construction. Heckmann [36] added the empty set to this construction and call it the small powerdomain. It is worth mentioning that Gunter considered as well the mixed powerdomain with the empty set added and he arrived exactly to the same results as Heckmann did.

Definition 2.5.32. *The **Mixed powerdomain** σD over an algebraic cpo D with coherent topology has carrier:*

$$\{(A^H, A^S) \mid (1)\, A^H \text{ is a non-empty Scott-closed set of } D \\ (2)\, A^S \text{ is a non-empty Scott-compact upper set of } D \\ (3)\ \text{there is a subset } A^M \text{ of } A^S \text{ with } A^H = \mathsf{cl}(A^M)\}.$$

Its order is inherited from ΣD. The sandwiches in σD are called tight.

The derived operations are tight.

- If A and B are tight, then $A \uplus B$ is tight.
- The result of the operation η is tight.
- If f generates tight sandwiches for all arguments, and A is tight, then $ext(f(A))$ is tight.

Again the equation for the derived operation *map* may be computed from the definition of *ext*

- $map = ext(\eta(f))$.

Chapter 3

Partial Real Numbers

3.1 The Unit Interval Domain

In this section, we present the results given by Scott [74] for the interval domain. We use as our mathematical interpretation of the programming language presented in Chapter 9, a suitable powerdomain of the interval domain.

Definition 3.1.1. *The* ***interval domain*** *is the set $\mathcal{R}$ of all non-empty compact subintervals of the Euclidean real line ordered by reverse inclusion*

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

It is easy to prove that from the domain theoretical point of view the interval domain is in fact a domain. Moreover if a bottom element is added to the interval domain which is denoted by the non-compact interval $(-\infty, \infty)$ then it becomes a bounded complete domain denoted by $\mathcal{R}_\perp$. A classical picture in the literature to represent $\mathcal{R}_\perp$ is given in Figure 3.1. Given $x \in \mathcal{R}$,

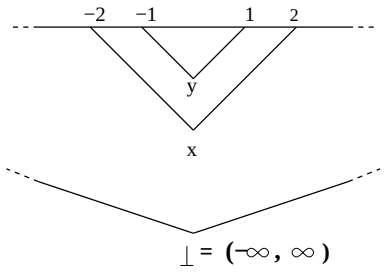


Figure 3.1: Interval Domain.

it is written as $x = [\underline{x}, \overline{x}] \subseteq \mathbb{R}$. Informally, $\underline{x}$ represent the infimum (inf) of the interval x and $\overline{x}$ the supremum (sup) of x.

$$\underline{x} = \inf x \quad \overline{x} = \sup x.$$

The least upper bounds in the interval domain are denoted by non-empty intersections of intervals.

Proposition 3.1.2. *A subset $A \subseteq \mathcal{R}$ has a least upper bound iff it has non-empty intersection; the lub is given by*

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

Greatest lower bounds are guaranteed to exist if lower bounds exist.

Proposition 3.1.3. *Any subset $A \subseteq \mathcal{R}$ with a lower bound has a greater lower bound, given by:*

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

The way-below relation on the domain $\mathcal{R}$ is defined as

Definition 3.1.4. *Let $x, y \in \mathcal{R}$, $x \ll y$ iff $\underline{x} < \underline{y}$ and $\overline{y} < \overline{x}$.*

From a topological point of view, we say that $x \ll y$ if and only if the interior of x contains y.

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

In the presentation of the programming language in Chapter 9, it is convenient to work with a compact subset of the Euclidean real line. Although the results are presented for a compact subset, it is shown in [24] how they can easily be extended for the whole real line.

In this section, we consider what is known as the unit interval domain denoted by $\mathcal{I} = \mathbf{I}[0, 1]$.

Definition 3.1.5. *The **unit interval domain** $\mathcal{I}$ is the set of all non-empty closed intervals contained in the interval $[0, 1]$ ordered by reverse inclusion.*

The unit interval domain is a bounded complete, countably based domain. The bottom element of $\mathcal{I}$ is the interval $[0, 1]$. Its way-below order is given by:

$$x \ll y \text{ iff } \underline{x} = 0 \text{ or } \underline{x} < \underline{y}, \text{ and } \overline{y} < \overline{x} \text{ or } \overline{y} = 1.$$

Remark. In Chapter 8 we use domains of the form $\mathcal{I}[-r,r]$; for $r \in \mathbb{Q}$. These domains are the sets of all non-empty closed intevals contained in the interval $[-r,r]$ ordered by reverse inclusion. These domains have the same properties that the unit interval domain.

There is another useful definition when we add $[-\infty,\infty]$ to the interval domain.

Definition 3.1.6. *The **extended real line** is the two-point compactification*[1] *of the real line, denoted as* $\mathbb{R}^* = [-\infty,\infty]$.

The set $\mathcal{R}^* = \mathbf{I}\mathbb{R}^*$ of all non-empty compact intervals of the extended real line is a bounded complete, countably based domain, referred to as the **extended interval domain**. Its way-below relation is given by:

$$x \ll y \text{ iff } \underline{x} = -\infty \text{ or } \underline{x} < \underline{y}, \text{ and } \overline{y} < \overline{x} \text{ or } \overline{y} = \infty.$$

3.2 Maps between elements of the interval domain

There is a homeomorphism between the real line and the set of maximal elements of the interval domain. The **singleton map** $s : \mathbb{R} \to \mathcal{R}$ given by $r \mapsto \{r\}$ is onto the set of maximal elements of $\mathcal{R}$ (denoted by $\mathsf{Max}\ \mathcal{R}$). In the other direction, the set $\uparrow\!\!\uparrow x$, for $x \in \mathcal{R}$, forms a basis of the Scott topology of $\mathcal{R}$, and since $\uparrow\!\!\uparrow x \cap \mathsf{Max}\ \mathcal{R} = \{\{r\} \mid \underline{x} < r < \underline{x}\} = \{s(r) \mid r \in (\underline{x},\overline{x})\}$, the set $\{s(r) \mid r \in (a,b)\}$ for open interval (a,b) forms a base of the relative topology on $\mathsf{Max}\ \mathcal{R}$.

$\mathsf{Max}\ \mathcal{I}$ and $\mathsf{Max}\ \mathcal{R}^*$ are also homeomorphic to the unit interval and the extended real line, respectively.

Definition 3.2.1. *An element $x \in \mathcal{R}$ is called a **partial real number** iff $\underline{x} \neq \overline{x}$, i.e. x is a non-singleton interval.*

In this sense, the interval domain is referred to as the **partial real line**. Similarly, the unit interval domain is referred to as the **partial unit interval** and the extended interval domain as the **extended partial real line**.

Proposition 3.2.2. *For every continuous map $f : \mathbb{R}^n \to \mathbb{R}$, there is a canonical extension $\mathbf{I}f : \mathcal{R}^n \to \mathcal{R}$ defined by:*

$$\mathbf{I}f(x_1 \ldots, x_n) = \{f(r_1, \ldots, r_n) \mid r_1 \in x_1, \ldots, r_n \in x_n\}.$$

[1] see [80, Page 92]

Proof. We can see using topological arguments that $\mathbf{I}f$ is well defined. For the case when $n = 1$, it is as follows: Since f is continuous, the image of a connected set x under f is connected. The same applies for compact sets. Hence it maps compact intervals to compact intervals. The same argument works for n arbitrary. □

Proposition 3.2.3. $\mathbf{I}f$ *is Scott continuous.*

Proof. See [24] □

It is easy to see that the canonical extension is not only the greatest monotone extension but also the greatest continuous extension.

If f is increasing in each argument with respect to its natural order of $\mathbb{R}$, then $\mathbf{I}f$ is given pointwise

$$\mathbf{I}f(x_1, \ldots, x_n) = [f(\underline{x}_1, \ldots, \underline{x}_n), f(\overline{x}_1, \ldots, \overline{x}_n)].$$

For f decreasing, then $\mathbf{I}f$ is given in the reverse order

$$\mathbf{I}f(x_1, \ldots, x_n) = [f(\overline{x}_1, \ldots, \overline{x}_n), f(\underline{x}_1, \ldots, \underline{x}_n)].$$

As an example, the canonical extension of the addition map is given by

$$x + y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}].$$

The extension property discussed above applies to the partial unit interval as well. For the extended partial real line, further considerations are needed [24] but the main point is that there is a greatest continuous extension.

Although there exists a greatest continuous extension, Escardó, Hofmann and Streicher [27] have shown that for simple arithmetic operations such as addition, a parallel function can be defined from it. Furthermore, they have shown that from any continuous extension of addition (not only the greatest one), parallel functions can be defined.

Because we want a sequential programming language, given the above we cannot take the interval domain as the model for our language. Instead, we consider a powerdomain of the interval domain, specifically the Hoare powerdomain as discussed in the introduction and as to be elaborated later. The reasons for considering the Hoare powerdomain and not other powerdomains are presented in Chapter 7. A sequential definition for addition is presented in Chapter 8.

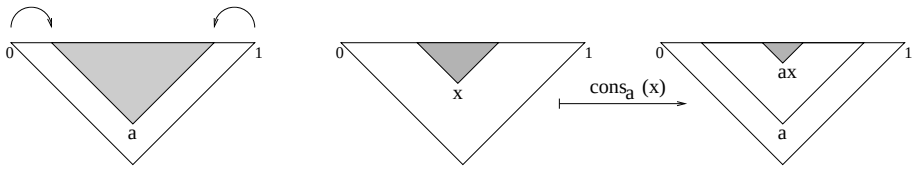


Figure 3.2: The map cons_a.

3.2.1 The maps cons and tail

Two maps play an important role in the operational semantics of the programming languages of Chapters 5 and 9, namely $\mathsf{cons}_a : \mathcal{I} \to \mathcal{I}$ and $\mathsf{tail}_a : \mathcal{I} \to \mathcal{I}$.

Notation 3.2.4. *For $x \in \mathcal{I}$, we write $\kappa_x = (\overline{x} - \underline{x})$; i.e., κ_x represents the diameter of x. In a similar way we write $\mu_x = \underline{x}$. In Chapter 10 we use $|x|$ instead of κ_x.*

cons_a is a continuous increasing affine map defined by:

$$\mathsf{cons}_a(x) = [(\kappa_a)\underline{x} + \mu_a, (\kappa_a)\overline{x} + \mu_a]$$

That is, given $x, a \in \mathcal{I}$, rescale and translate the unit interval so that it becomes a, and define $\mathsf{cons}_a(x)$ to be the interval which results from applying the same rescaling and translation to x. Figure 3.2 illustrates the behaviour of the map cons_a. We sometimes write $\mathsf{cons}_a(x)$ as ax. Later on, multiplication will be signified by $\times$ to avoid any confusion.

It is immediately clear that ax is a subinterval of a. The rescaling factor is κ_a, and the translation constant is the left end-point of a. If a is maximal, then its diameter is zero, so that $ax = a$. We have that $\mathsf{cons}_a \circ \mathsf{cons}_b = \mathsf{cons}_{ab}$ where ab is defined as above.

Observation. In the definitions presented in Chapter 8, we sometimes work with the continuous increasing map $\mathsf{cons}_a : \mathcal{I}[-r, r] \to \mathcal{I}[-r, r]$ defined as:

$$\mathsf{cons}_a(x) = \left[\left(\frac{\kappa_a}{2r}\right)\mu_x + \left(\frac{\overline{a} + \underline{a}}{2}\right), \left(\frac{\kappa_a}{2r}\right)\overline{x} + \left(\frac{\overline{a} + \underline{a}}{2}\right)\right]$$

where $r \in \mathbb{Q}$.

We would like a continuous left inverse tail_a of the map cons_a, in the sense that

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

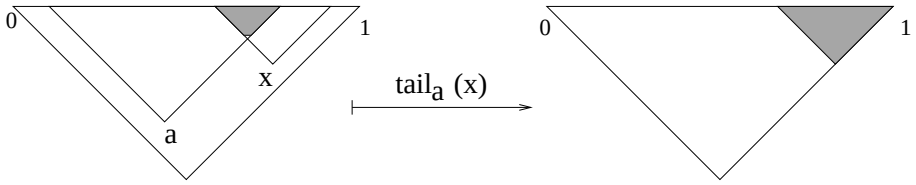


Figure 3.3: The map tail_a.

However, in general there is no such map (see [24], page 88, for a detailed explanation). We define for every non-maximal $a \in \mathcal{I}$ a continuous left inverse tail_a of the map cons_a.

Definition 3.2.5. *For every non-maximal $a \in \mathcal{I}$, the continuous map $\mathsf{tail}_a : \mathcal{I} \to \mathcal{I}$ is defined as*

$$\mathsf{tail}_a(x) = [(x' - \underline{a})/\kappa_a, 1 - (\overline{a} - y')/\kappa_a]$$

where

$$x' = \min(\max(\underline{a}, \underline{x}), \overline{a})$$

$$y' = \max(\min(\overline{a}, \overline{x}), \underline{a})$$

Figure 3.3 may give a better intuition of the behaviour of this map.
Consider two intervals $x, y \in \mathcal{I}$, with the following conditions:

1. $\underline{x} \neq \overline{x}$,
2. $x \sqsubseteq y$;

then there exists a unique $c \in \mathcal{I}$ such that $xc = y$ which we denote by $y \setminus x$. In fact:

$$c = [(\underline{y} - \mu_x)/\kappa_x, (\overline{y} - \mu_x)/\kappa_x].$$

3.3 The Hoare Powerdomain of the Interval Domain

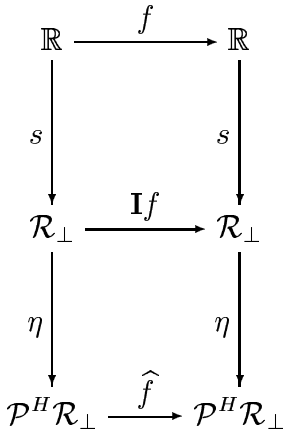
Given that we will use the Hoare powerdomain as the mathematical meaning of our programming language as explained above, and because the interval

domain forms a domain, all the properties stated in Section 2.5.4 for the Hoare powerdomain hold when applied to the interval domain. Moreover, they are applicable to the unit interval domain and the extended interval domain. We now summarize these properties, for the sake of quick reference.

1. $\mathcal{P}^H\mathcal{R}_\perp$ forms a complete lattice,
2. if $X, Y \in \mathcal{P}^H\mathcal{R}_\perp$, then $X \uplus Y = X \cup Y \in \mathcal{P}^H\mathcal{R}_\perp$,
3. if $x \in \mathcal{R}_\perp$ then $\eta(x) = {\downarrow}x \in \mathcal{P}^H\mathcal{R}_\perp$,
4. if $A \subseteq \mathcal{P}^H\mathcal{R}_\perp$ then
$$\bigsqcup A = \mathsf{cl}\left(\bigcup_{A_i \in A} A_i\right),$$
5. if D is a domain and $f : \mathcal{R}_\perp \to \mathcal{P}^H D$ is a continuous map, then there exists a unique homomorphism $\overline{f} : \mathcal{P}^H\mathcal{R}_\perp \to \mathcal{P}^H D$ given by $A \mapsto \mathsf{cl}(\bigcup_{a \in A} fa)$.
6. if $f : \mathcal{R}_\perp \to \mathcal{R}_\perp$ is a continuous map, then there is a unique homomorphism $\widehat{f} : \mathcal{P}^H\mathcal{R}_\perp \to \mathcal{P}^H\mathcal{R}_\perp$ given by $A \mapsto \mathsf{cl}f[A]$.

Proposition 3.3.1. *If $f : \mathbb{R} \to \mathbb{R}$ is a continuous map, then there is a continuous map $\widehat{f} : \mathcal{P}^H\mathcal{R}_\perp \to \mathcal{P}^H\mathcal{R}_\perp$.*

Proof. By Proposition 3.2.3, for any continuous $f : \mathbb{R} \to \mathbb{R}$, there is a continuous extension $\mathbf{I}f : \mathcal{R}_\perp \to \mathcal{R}_\perp$. By 6 above, there is a continuous extension of $\mathbf{I}f$, $\widehat{f} : \mathcal{P}^H\mathcal{R}_\perp \to \mathcal{P}^H\mathcal{R}_\perp$. In other words, the following diagram commutes:



□

Given that cons_a and tail_a are continuous maps (see [24, Chapter 9]) by Theorem 2.5.9 of Section 2.5, there are continuous extensions $\widehat{\mathsf{cons}}_a$ and $\widehat{\mathsf{tail}}_a$ such that the following diagrams commute:

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathsf{cons}_a} & \mathcal{I} \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \mathcal{P}^H\mathcal{I} & \xrightarrow[\widehat{\mathsf{cons}}_a]{} & \mathcal{P}^H\mathcal{I} \end{array} \tag{3.1}$$

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathsf{tail}_a} & \mathcal{I} \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} \\ \mathcal{P}^H\mathcal{I} & \xrightarrow[\widehat{\mathsf{tail}}_a]{} & \mathcal{P}^H\mathcal{I} \end{array} \tag{3.2}$$

These extensions are used in the recursive definitions presented in Chapter 8.

Chapter 4

Real-number Computation

This is a summary of the discussions of [69, 71, 26]. There are several approaches to real number computation; among these are: floating point arithmetic, interval analysis, stochastic rounding, symbolic manipulation and exact real arithmetic. Probably the best known and most used approach is floating point arithmetic. Now we briefly discuss each approach and concentrate on the exact real-number arithmetic which is the approach used in this work.

4.1 Floating point arithmetic

Floating point arithmetic has been the standard approach to representing real numbers in a computer. There is a huge range of material in which several operations using this kind of representation have been intensively studied. There is a standard commonly used representation, the IEEE-754, in which real numbers can be represented in two ways: 32-bit 'single' precision (see Figure 4.1) and 64-bit 'double' precision. Of course, there are other more unfamiliar representations, such as VAX floating point, which vary from IEEE-754 in the format in which a real number is stored.

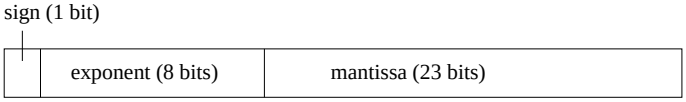


Figure 4.1: The mantissa in a 32-bit 'single' precision is represented by 23 bits and the exponent by 8 bits. The remaining bit is used for the sign.

As Figure 4.1 shows, both the mantissa and the exponent are of fixed size; hence only a finite subset of numbers in the whole real line can be represented.

This set is called the set of machine-representable reals. Moreover only a finite number of the elements can be represented exactly.

A major problem with this representation is the inaccuracy of the results. For example, if x and y are two real numbers in the set of representable reals, there are operations such as multiplication or addition which do not necessarily produce an element in the set of representable reals.

This situation could be described as partially overcome by rounding off the answer of the computation to the nearest representable real every time an operation is performed. However, this rounding off generates an error which produces serious effects on the accuracy of the result.

Mathematically, this problem is formalised by showing that in floating-point computation, the representable reals fail to form a field by not being closed under the field operations. On the other hand, it is well known that the whole real line forms a field.

This problem has been intensively investigated and one possible solution to deal with it has been the use of error analysis (numerical analysis deals with this subject).

4.2 Interval Arithmetic

In interval arithmetic, a real number is represented by a pair of numbers which denote an interval containing the real number in question [60]. A pair of rational numbers or floating-point numbers can be used to represent such an interval.

Each time an operation is performed, new intervals are computed in order to obtain the desired property. If, for example, floating-point arithmetic with error analysis is used, the interval computed can be calculated by rounding the upper bound of the interval strictly upwards and the lower bound strictly downwards.

The usefulness of interval arithmetic resides in the fact that once a programming language capable of doing operations using the interval approach has been developed, there is no need to analyse each computation as in the case of floating-point arithmetic.

Once the boundaries of the result are known, it is possible to combine them with respect to the real solution to express the result in terms of the number of correct significant digits of the interval.

Furthermore, if for a given input only certain digits representing a real number are known (e.g. a physical measurement of some kind), the output result would represent and reflect this in the compactness of the bounds of the interval.

As in floating-point arithmetic with error analysis, the problem with interval arithmetic is that it does not bring the solution closer to exactness. Although there is an interval which is known to contain the exact solution, the bounds of this interval can be too wide apart to provide a useful answer.

4.3 Stochastic Rounding

A difference of floating-point arithmetic which is rounded off to the nearest representable number and to interval arithmetic where the bounds of the interval are rounded off to the nearest representable numbers above and below, the stochastic approach rounds off numbers at each stage using stochastic techniques [14, 82].

These techniques are applied to the same computation several times to obtain the final result. Probability theory is applied to estimate the correct result using stochastic methods.

In contrast to interval arithmetic, which guarantees an accurate result (at least it is known how many digits of the result are correct), this approach does not. However in most cases a better result is obtained using stochastic methods than by floating-point arithmetic. These are reliable, so long as the calculation can be made with a certain probability using this approach.

4.4 Symbolic Approaches

In the symbolic approaches, instead of doing computations with real numbers, expressions are manipulated in terms of symbols representing variables, constants and functions.

Manipulating an expression entirely in terms of symbols guarantees an exact representation of the correct result. For example, in symbolic approaches, integration can be presented given a symbolic expression which represents the answer exactly.

There are certain arithmetical tasks for which numerical approaches cannot produce a result as accurate as that which a symbolic expression can provide. For example, the simplification of a symbolic expression may result in the knowledge of some useful properties not considered using numerical methods. Consider

$$4\mathsf{arctan}\left(\mathsf{sin}^2\left(\frac{1}{23}\right)+\mathsf{cos}^2\left(\frac{1}{23}\right)\right).$$

If this expression is simplified, the result is that it is equal to π. However if a

computation of this expression is carried out using floating-point arithmetic, the result produced would not be as accurate as might be necessary.

Despite the usefulness of symbolic approaches, for certain applications, they cannot replace numerical approaches. For instance, although it may be not known that the symbolic expression represents an exact result, it might not be possible to reduce the expression any more (not at least with the methods available nowadays). For this reason, an evaluation of the expression is required. Any of the approaches described above can be used for the evaluation of the expression; however, the result obtained inherits the error produced by each.

In the next section, we discuss exact real number arithmetic. Hence symbolic approaches and exact real number arithmetic can be combined to produce exact results in a general framework.

4.5 Exact-Real Number Arithmetic

Exact real arithmetic is a numerical method for real number computation, in which a real number is represented by infinite data structures such as streams [83, 63, 70, 71]. Among its main features is the reliability of the produced results. It solves the problem of inaccuracy in floating-point arithmetic and the uncertainty of the distance between the bounds of intervals in interval arithmetic. In addition, there are applications in which symbolic approaches are not effective but exact-real arithmetic is.

In exact real arithmetic, a result can generally be calculated to any degree of accuracy, however inefficiency in the calculations characterises this technique. As stated by Plume [69], the manipulation of infinite data structures has an important role in this respect since they are inherently expensive to manage, in contrast with the type of fixed sized data structures used in floating-point arithmetic.

An important aspect of exact real number computation is that, despite the infinite representations of real numbers, in a calculation, a finite number of data in the output result relies on a finite amount of information from the input. However, in practice the required finite amount of input can be extremely large, resulting in inefficient computations.

Despite the inefficiency in exact real number computation, there have been a number of theoretical and practical attempts to find a viable framework for exact real number arithmetic, in which a efficient number of approaches have been proposed.

Before describing the different approaches to exact real number computation in detail, we present a problem of representation discovered by

Brouwer [12] in 1920, which indicates that infinite decimal numerals are not a suitable representation for exact computation.

4.5.1 The Problem of representation

In exact real arithmetic as previously mentioned, we require a data type with infinite data to represent real numbers. If for the moment we adopt a representation as an infinite sequence of decimal numbers (a decimal number is an element in the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$), we immediately notice that for any infinite sequence such as 0.33333... there is no least significant digit. This is because the representation extends infinitely to the right. In this sense, operations performed between real numbers in this arithmetic should be computed from left (the most significant digit) to right (the least significant digit), in contrast with floating-point arithmetic, where most of the operations are performed in the opposite direction - from the least significant digit to the most significant. For example, addition in floating-point arithmetic is performed by first adding the two least significant digits of the inputs and the process is continued to the left.

Reasoning about possible representations of the way in which operations are implemented in the desired left-to-right order, it turns out that binary or decimal representations are not suitable for exact real number computation. This fact can be easily illustrated by providing basic arithmetic operations such as addition or subtraction which are not computable if binary or decimal representations are used.

In general, it is not possible to determine a single digit of the output without examining infinitely many digits of the input. This fact clashes with the feature of exact real arithmetic that finitely many digits of the output depend on finitely many digits from the input. Considering examples similar to the previous one, we can argue that any integer base suffers from the same problem.

A topological proof of this fact can be seen as follows. For simplicity, we consider only fractional numbers, ignoring the decimal point and the leading zero in decimal notation. A *numeral* is an infinite sequence over the digit alphabet $D = \{0, 1, \ldots, 9\}$. A numeral $\alpha \in D^{\mathbb{N}}$ denotes the number

$$[\![\alpha]\!] = \sum_{i \geq 0} \alpha_i \cdot 10^{-(i+1)} \in [0, 1].$$

The map $\alpha \mapsto [\![\alpha]\!]$ is called the **denotation map**, it is the surjection

$$q : D^{\mathbb{N}} \twoheadrightarrow [0, 1].$$

However, it is not an injection because the decimal rationals $m/10^n \in (0,1)$ have two decimal notations - the reason is that we consider infinitely long runs of digit 9 as legitimate as infinitely long runs of digit 0, because there is no way of ruling out the former by computational means.

Definition 4.5.1. *A function $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$* ***realizes*** *a function $f : [0,1] \to [0,1]$ if*

$$f([\![\alpha]\!]) = [\![\phi(\alpha)]\!],$$

as illustrated the following diagram:

$$\begin{array}{ccc} D^{\mathbb{N}} & \xrightarrow{\ \phi\ } & D^{\mathbb{N}} \\ {\scriptstyle q}\downarrow & & \downarrow{\scriptstyle q} \\ [0,1] & \xrightarrow{\ f\ } & [0,1]. \end{array}$$

The main interest in this work resides on *computable* realizers. A realizer lies at the operational level (in the set of rules of the programming language considered) while the function which it realizes lies at the denotational level (the mathematical meaning). As stated before, one necessary (but not sufficient) condition for a function $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$ to be computable is that finite sequences of $\phi(\alpha)$ depend only on finite sequences of α. In this case we say that ϕ is of *finite character.*

Proposition 4.5.2 (Escardó [25]). *A function $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$ is of finite character iff it is continuous.*

This proposition is the link between computability and continuity at the operational level. To get a link at the level of mathematical meaning, we state the following.

Proposition 4.5.3 (Escardó [25]). *The surjection $q : D^{\mathbb{N}} \twoheadrightarrow [0,1]$ is a topological quotient map.*

Proof. It is proved by the fact that a continuous surjection of compact Hausdorff spaces is always a quotient map. □

By a basic property of topological quotients [80, 44], we get the following.

Corollary 4.5.4 (Escardó [25]). *A function $f : [0,1] \to [0,1]$ with a realizer $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$ of finite character is necessarily continuous.*

Unfortunately, however, the converse is not true.

Proposition 4.5.5 (Escardó [25]). *There are continuous functions* $f : [0,1] \to [0,1]$ *with no realizer* $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$ *of finite character, for example* $f(x) = 3x/10$.

Proof. Let $\phi : D^{\mathbb{N}} \to D^{\mathbb{N}}$ be a realizer of f. Then for any α and any $n \geq 0$, the value of $\phi(3^n 2\alpha)$ is of the form 0β. Thus, if ϕ were continuous, $\phi(3^\omega)$ would be of the form $0\beta'$. But similarly, for any α and $n \geq 0$, the value of $\phi(3^n 4\alpha)$ is of the form 1β, and the continuity of ϕ would imply that $\phi(3^\omega)$ would be of the form $1\beta''$. Thus, $\phi(3^\omega)$ would have to be of the form $0\beta'$ and $1\beta''$ at the same time. Since this is impossible, we conclude that ϕ is not continuous. □

The results presented above are independent of the base; but different counter-examples are needed for different bases.

To obtain algorithms for exact real number arithmetic using infinite objects, a different representation has to be used. In the next section we describe what we believe is the simplest representation of real number arithmetic. In further sections, we move on to discussing different and more elaborated frameworks. It is worth noting that all the representations used in exact real number arithmetic are equivalent to each other, in the sense that it is possible to effectively translate one representation into another. One of the main reasons to consider different frameworks is our concern with efficiency aspects. In this sense one representation can in some applications be more efficient than the others.

4.5.2 Signed Digit Representations

Signed digit representation can be used to define algorithms to compute basic arithmetic operations, and also higher level transcendental functions. A main characteristic using this kind of representation is the high degree of redundancy on operations (the infinite character of sequences is the main reason for this fact).

Although we present signed digit representation, the theory is easily translated to any other integer base B with sign. In addition, Di Gianiantonio [16] presented a notation called the golden ratio which consists of a non integer base with digits 0 and 1, in which we can translate effectively between it and any other notation for exact real number computation. This means that the golden ratio notation is as expressive as any other representation. Golden ratio notation has been implemented by McGaw [58].

The following is taken from Escardó [25]. A signed-digit numeral is an infinite sequence over the signed digit alphabet $\mathbf{3} = \{\overline{1}, 0, 1\}$, where $\overline{1}$ stands for -1. An arrow above a digit is used to denote an infinite sequence of that digit. For example $\vec{1}$ denotes an infinite sequence of ones. A numeral $\alpha \in \mathbf{3}^{\mathbb{N}}$ denotes the number

$$[\![\alpha]\!] = \sum_{i \geq 0} \alpha_i \cdot 2^{-(i+1)} \in [-1, 1].$$

Formally, the surjection $\alpha \mapsto [\![\alpha]\!]$ is a quotient map $q : \mathbf{3}^{\mathbb{N}} \twoheadrightarrow [-1, 1]$. This representation is also known as a redundant binary representation of $\mathbb{R}$ [45]. There is a unique representation for minus one, $\vec{-1}$, and for one, $\vec{1}$, and infinitely many representations for all the other numbers in this range.

The crucial result using this kind of representation, in contrast to Proposition 4.5.5, is the following:

Proposition 4.5.6. *Every continuous function $f : [-1, 1] \to [-1, 1]$ has a realizer $\phi : \mathbf{3}^{\mathbb{N}} \to \mathbf{3}^{\mathbb{N}}$ of finite character.*

Proof. See Müller [61] and Weihrauch and Kreitz [87, 53]. □

The same holds for functions of several arguments with a realizer defined in the obvious way.

For the purpose of this section, a function $f : [-1, 1] \to [-1, 1]$ is computable if it has a computable realizer $\phi : \mathbf{3}^{\mathbb{N}} \to \mathbf{3}^{\mathbb{N}}$. The question at this point would be how computability on $\mathbf{3}^{\mathbb{N}}$ is to be defined. According to [46, 86, 85] the theory of computability using Turing machines can be used for this purpose.

Finally, as stated above, computable functions $f : [-1, 1] \to [-1, 1]$ are continuous. More generally, a computable partial function is continuous in its domain of definition.

4.5.3 Infinite Sequences of Linear Maps

This approach was proposed by Avizienis [5] and is considered in several works, such as Watanuki and Eroegovac [84], Boehm and Cartwright [10], Di Gianantionio [17], Escardó [22], Nielsen and Kornerup [63] and Ménissier-Morain [59]. In this approach, the main concept is an increasing linear map, which is a function of the form

$$f(x) = ax + b$$

where a and b can be rational or dyadic numbers (a dyadic number is of the form $m/2^n$ for m, n integers).

We recall from Chapter 3 that an extended real number is an element of the interval $[-\infty, \infty]$. In this sense, an element $r \in [-\infty, \infty]$ is given by the intersection of a nested sequence of rational (dyadic) intervals obtained by applying an infinite sequence of compositions of increasing linear maps to the base interval $[-\infty, \infty]$.

$$\{r\} = \bigcap_{n \geq 1} f_1 f_2 f_3 \cdots ([-\infty, \infty]).$$

Arithmetic operations and transcendental functions have already been defined in this approach. A programming language using this representation has been given by Escardó [22]. Escardó has shown that all computable functions defined in the interval domain model are definable in the programming language. However certain parallel operators are needed for this purpose. We present this programming language in Chapter 5 as it will be the main reference throughout this work. Also, Di Gianantonio [18] uses this approach to show some sequentiality results in exact real number computation.

4.5.4 Continued Fraction Expansions

This approach was proposed by Gosper [7] and developed by Jones [65] and Vuillemin [83]. Lester [54] has presented an implementation of this approach. In recent years, Kornerup and Matula [48, 49, 50, 52] have presented more results in this field.

A real number r in the continued fraction approach is represented by a stream $[d_0, d_1, d_2, \ldots, d_i, \ldots]$ of integers such that:

$$r = \lim_{i \to \infty} d_0 + \cfrac{1}{d_1 \cfrac{1}{d_2 + \cfrac{1}{\ddots + \frac{1}{d_i}}}}$$

One of the main advantages of this representation with respect to signed-digit representation is that certain real numbers are easier to represent by the former than by the latter. For example:

$$\phi = \frac{1 + \sqrt{5}}{2} = [1, 1, 1, 1, 1, 1, \ldots].$$

$$e = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, \ldots].$$

4.5.5 Infinite Composition of Möbius Transformations

Vuillemin [83] showed that Möbius transformations generalise the other two approaches described before. Quasi-normalized floating points [84] can be represented using this framework as shown by Nielsen and Kornerup [63]. Edalat and Potts [20, 71] have presented efficient algorithms for this representation. A semantics of exact real number computation using this representation is presented in [72].

In this approach (also known as **linear fractional transformations**) transformations have the form:

1.
$$t(x) = \frac{ax + c}{bx + d},$$

2.
$$t(x, y) = \frac{axy + cx + ey + g}{bxy + dx + fy + h},$$

where a,b,c,d,e,f,g and h are integers. Transformations of the first form are known as one-dimensional linear fractionals (lfts). The integer coefficient of these transformations are arranged conveniently in the 2×2 matrix

$$M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}.$$

Transformations of the second form are known as two dimensional lfts. The integer coefficient of these transformations are arranged conveniently in the 2×4 tensor

$$M = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}.$$

The composition of either of these can be seen as the multiplication of matrices or tensors.

A non-negative extended real number in this approach is an element of $[0, \infty]$. A non-negative extended real number r is given by the intersection of a nested sequence of rational intervals obtained by applying an infinite sequence of composition of matrices (as described above) applied to the base interval $[0, \infty]$

$$\{r\} = \bigcap_{n \geq 0} M_0 \cdot M_1 \cdot M_2 \cdots ([0, \infty]).$$

Arithmetic operations and transcendental functions are developed in this approach using infinite multiplications of tensors, analogous to the multiplication of matrices to represent real numbers [20]. One of the main advantages of this representation is that most of the algorithms in the continued fraction approach can be translated to this setting almost directly. Many efficient algorithms have been implemented using this technique; however there is no general consensus that this technique is better than the others. Heckmann [38, 37] has shown that a problem with this approach lies in the inefficiency regarding space.

Chapter 5

The Programming Languages PCF and Real PCF

In this chapter, we introduce the language PCF given by Plotkin [68] and its extension to real numbers given by Escardó [24]. In Section 5.1, we introduce the language PCF. As we have emphasised in the introduction of the thesis, we want to define a sequential programming language for exact real number computation. In this sense PCF will be the kernel of our language. In Section 5.2, we present the programming language Real PCF, the main reason to introduce Real PCF in this thesis is that the programming language *LRT* presented in Chapter 9 can be seen as the programming language Real PCF with two important substitutions:

- The constructor head_r explained below is replaced by the constructor $\mathsf{rtest}_{a,b}$ described in Chapter 7.
- The parallel conditional pif is replaced by the sequential conditional if.

Although this variation of the language seems to be at first glance simple and straightforward, it brings a new theory to study, which is one of the main purposes of this thesis.

5.1 The Programming Language PCF

PCF is a typed lambda calculus language extended with some primitive operations to deal with expressions involving natural numbers and booleans. The lambda calculus [6] is a paradigm for functional programming which has been thoroughly investigated. We assume previous knowledge of lambda calculus and present the basic definitions of PCF. A call-by-name evaluation strategy of the language is presented.

5.1.1 Syntax

The language PCF is a typed language whose set *Type* of type expressions T is defined by the grammar:

$$\sigma ::= \texttt{nat} \mid \texttt{bool} \mid \sigma \to \tau$$

where σ, τ are metavariables ranging over the set of types; `nat` and `bool` are the type constants for naturals and booleans respectively. The types `nat` and `bool` are called ground types.

Let $(\sigma_1, \ldots, \sigma_n, \tau)$ stands for $(\sigma_1 \to (\sigma_2 \to, \cdots (\sigma_n \to \tau) \cdots))$ meaning that applications of functions of one argument can be viewed as functions of several variables by uncurrying. Given a collection $\mathcal{L}$ of formal constants, each having a fixed type, and a family of formal variables $\{\alpha_i^\sigma\}(i \neq 0)$ for each type σ, the $\mathcal{L}$-terms are given by the following inductive rules:

1. Every variable $\{\alpha_i^\sigma\}$ is an $\mathcal{L}$-term of type σ.
2. Every constant c of type σ is an $\mathcal{L}$-term of type σ.
3. If M and N are $\mathcal{L}$-terms of types $(\sigma \to \tau)$ and σ respectively then (MN) is an $\mathcal{L}$-term of type τ.
4. If M is an $\mathcal{L}$-term of type τ then $(\lambda\alpha_i^\sigma M)$ is an $\mathcal{L}$-term of type $\sigma \to \tau$.

When no confusion arises the superscript σ of α will be omitted. The application of terms (MN) is understood to be associative to the left. When $\mathcal{L}$ is understood from the context it is not used as a prefix. The fact that a term M has type variable σ is denoted by $M : \sigma$.

The set of *free variables* of a term M, denoted by $FV(M)$, is inductively defined by

1. $FV(\alpha_i^\sigma) = \{\alpha_i^\sigma\}$
2. $FV(c) = \emptyset$
3. $FV(MN) = FV(M) \cup FV(N)$
4. $FV(\lambda\alpha_i^\sigma M) = FV(M) - \{\alpha_i^\sigma\}$

A term is *closed* if $FV(M) = \emptyset$. *Programs* are closed terms of ground type.

The language $\mathcal{L}_{PCF}$ consists of the $\mathcal{L}$-terms and the following constants:

- `true` : `bool`, `false` : `bool`,

- $\mathtt{k}_n : \mathtt{nat}$ for each natural number n,
- $(+\mathtt{1}) : \mathtt{nat} \to \mathtt{nat}$, $(-\mathtt{1}) : \mathtt{nat} \to \mathtt{nat}$, $(=\mathtt{0}) : \mathtt{nat} \to \mathtt{bool}$,
- $\mathtt{if} : (\mathtt{bool}, \sigma, \sigma, \sigma)$ for σ ground,
- $\mathtt{Y}_\sigma : (\sigma \to \sigma) \to \sigma$ for each σ.

5.1.2 Operational Semantics

The operational semantics is given by an *immediate reduction relation* $\to$, between terms of the language. It is defined by the following set of reduction rules.

1. constants for natural numbers:

$$(+\mathtt{1})\mathtt{k}_n \to \mathtt{k}_{n+1},$$

$$(-\mathtt{1})\mathtt{k}_{n+1} \to \mathtt{k}_n,$$

2. constants for booleans:

$$(=\mathtt{0})\mathtt{k}_0 \to \mathtt{true},$$

$$(=\mathtt{0})\mathtt{k}_{n+1} \to \mathtt{false},$$

3. sequential conditional:

$$\mathtt{if\ true}\ M\ N \to M,$$

$$\mathtt{if\ false}\ M\ N \to N,$$

4. fixed point:

$$\mathtt{Y}_\sigma M \to M(\mathtt{Y}_\sigma M),$$

5. Application:

$$(\lambda\alpha M)N \to [N/\alpha]M,$$

$$\frac{M \to M'}{M(N) \to M'(N)},$$

$$\frac{N \to N'}{M(N) \to M(N')}, \quad \text{for } M \in \{(+\mathtt{1}), (-\mathtt{1}), (=\mathtt{0}), \mathtt{if}\}.$$

The reduction relation preserves types, in the sense that if $M \to M'$ and M has type σ, so does M'.

We define $\to^*$ as the transitive closure of the relation $\to$. Also, we define evaluation from programs to constants as a partial function given by:

$$\texttt{Eval}(M) = c \text{ if } M \to^* c \text{ for some constant } c.$$

5.1.3 Denotational Semantics

The denotational semantics for $\mathcal{L}_{PCF}$ is given using the set of domains

$$\bigcup_{\sigma} D_\sigma \text{ for } \sigma \in T,$$

where

$$D_{\texttt{nat}} = \mathbb{N}_\perp, \quad D_{\texttt{bool}} = \mathbb{T}_\perp, \quad D_{\sigma\to\tau} = [D_\sigma \to D_\tau].$$

The semantics interpretation function $\mathcal{A}$ has the form

$$\mathcal{A} : \mathcal{L}_{PCF} \to Env \to \bigcup_{\sigma} D_\sigma,$$

where Env is the set of environments. An environment is a function ρ from the set of variables to $\bigcup_\sigma D_\sigma$ satisfying the condition

$$\rho(x^\sigma) \in D_\sigma.$$

The definition of $\mathcal{A}$ is by structural induction on lambda terms, we use the brackets $[\![\ ]\!]$ to distinguish operational and denotational meanings:

$$\begin{aligned}
\mathcal{A}[\![c]\!]_\rho &= \mathcal{B}[\![c]\!], \\
\mathcal{A}[\![x^\sigma]\!]_\rho &= \rho(x^\sigma), \\
\mathcal{A}[\![M^{\sigma\to\tau} N^\sigma]\!]_\rho &= \mathcal{A}[\![M^{\sigma\to\tau}]\!]_\rho(\mathcal{A}[\![N^\sigma]\!]_\rho), \\
\mathcal{A}[\![\lambda\alpha^\sigma M^\tau]\!]_{(\rho)(x)} &= \mathcal{A}[\![M^\tau]\!]_{\rho[x/\alpha^\sigma]} \text{ (with } x \in D_\sigma).
\end{aligned}$$

The function $\mathcal{B}$ for the interpretation of constants is defined as:

$$\mathcal{B}[\![\mathtt{K}_n]\!] = n.$$

$$\mathcal{B}[\![(+1)]\!](n) = \begin{cases} n+1, & \text{if } n \in \mathbb{N}; \\ \bot, & \text{if } n = \bot. \end{cases}$$

$$\mathcal{B}[\![(-1)]\!](n) = \begin{cases} n-1, & \text{if } n \in \mathbb{N} \text{ and } n \geq 1; \\ 0, & \text{if } n = 0; \\ \bot, & \text{if } n = \bot. \end{cases}$$

$$\mathcal{B}[\![(=0)]\!](n) = \begin{cases} \mathtt{true}, & \text{if } n = 0; \\ \mathtt{false}, & \text{if } n > 0; \\ \bot, & \text{if } n = \bot. \end{cases}$$

$$\mathcal{B}[\![\mathtt{if}]\!](b)(x)(y) = \begin{cases} x, & \text{if } b = \mathtt{true}; \\ y, & \text{if } b = \mathtt{false}; \\ \bot, & \text{if } b = \bot. \end{cases}$$

$$\mathcal{B}[\![\mathtt{Y}]\!](f) = \bigsqcup_{n \in \mathbb{N}} \{f^n(\bot)\}.$$

For a closed term M, its denotational semantics does not depend on the environment, in the sense that $\mathcal{A}[\![M]\!]_\rho = \mathcal{A}[\![M]\!]_{\rho'}$ for all ρ and ρ'.

Notation 5.1.1. *We let $[\![M]\!]$ stand for the denotation $\mathcal{A}[\![M]\!](\bot)$ of a closed term M with respect to an implicit semantics $\mathcal{A}$. Additionally for any term M, we let $[\![M]\!]_\rho$ stand for $\mathcal{A}[\![M]\!]_\rho$.*

In the following definition, α has to be chosen as some variable of appropiate type in each instance.

Definition 5.1.2. *Define terms Ω_σ by*

$$\Omega_\sigma = \mathtt{Y}_\sigma(\lambda\alpha^\sigma\alpha^\sigma)$$

for σ ground and

$$\Omega_{\sigma\to\tau} = \lambda\alpha^\sigma\Omega_\tau,$$

and define terms $\mathtt{Y}_\sigma^{(n)}$ by induction on n by

$$\mathtt{Y}_\sigma^{(0)} = \lambda\alpha^{(\sigma\to\sigma)}.\Omega_\sigma$$

$$\mathtt{Y}_\sigma^{(n+1)} = \lambda\alpha^{(\sigma\to\sigma)}.\alpha^{(\sigma\to\sigma)}(\mathtt{Y}_\sigma^{(n)}\alpha^{(\sigma\to\sigma)}).$$

Then $[\![\mathtt{Y}_\sigma]\!] = \bigsqcup_n [\![\mathtt{Y}_\sigma^{(n)}]\!]$ for any standard interpretation.

Definition 5.1.3. *The **syntactic information order** $\preccurlyeq$ is the least relation between terms such that*

1. $\Omega_\sigma \preccurlyeq M : \sigma$ *and* $\mathtt{Y}_\sigma^{(n)} \preccurlyeq \mathtt{Y}_\sigma$ *for all* σ,
2. $M : \sigma \preccurlyeq M : \sigma$, *and*
3. *if* $M \preccurlyeq M' : \sigma \to \tau$ *and* $N \preccurlyeq N' : \sigma$ *then* $(\lambda\alpha N) \preccurlyeq (\lambda\alpha N')$ *and* $MN \preccurlyeq M'N'$.

5.1.4 Computational Adequacy

The operational and denotational semantics are related by what is called the **adequacy property** which is proved in two stages.

Theorem 5.1.4 (Soundness). *If* $\mathtt{Eval}(M) = c$, *then* $[\![M]\!] = [\![c]\!]$

Proof. See [68] □

Theorem 5.1.5 (Completeness). *If* $[\![M]\!] = [\![c]\!]$, *then* $\mathtt{Eval}(M) = c$

Proof. See [68] □

Theorem 5.1.6 (Computational Adequacy of PCF). *For any program* $M \in \mathcal{L}_{PCF}$ *and constant* c,

$$\mathtt{Eval}(M) = c \text{ iff } [\![M]\!] = [\![c]\!].$$

Proof. Theorems 5.1.4 and 5.1.5. □

5.2 Real PCF

In order to represent real numbers in Real PCF, it is sufficient to implement one of the approaches to exact real number computation described in Section 4.5. In [23] infinite sequences of linear maps are used, however any other approach can be used.

5.2.1 Syntax

We let $\mathcal{L}^I_{PCF}$ denote the extension of $\mathcal{L}_{PCF}$ with a new ground type $\mathtt{I}$ for real numbers and suitable constants for real-number computation:

$$\begin{array}{l}\mathtt{cons}_a : \mathtt{I} \to \mathtt{I},\\ \mathtt{tail}_a : \mathtt{I} \to \mathtt{I},\\ \mathtt{head}_r : \mathtt{I} \to \mathtt{bool},\\ \mathtt{pif}_{\mathtt{I}} : (\mathtt{bool}, \mathtt{I}, \mathtt{I}, \mathtt{I}),\end{array}$$

for each non-bottom element $a \in \mathcal{I}$ with distinct rational end-points, so that the interpretation of $\mathtt{tail}_a$ given below is well-defined according to Section 3.2.1, and each rational $r \in (0,1)$. We refer to the ground type $\mathtt{I}$ as the **real number type**, and programs of real number type as **real programs**.

5.2.2 Operational Semantics

The reduction relation $\to$ of $\mathcal{L}_{PCF}$ is extended to $\mathcal{L}^I_{PCF}$ by the following rules:

1. $\mathtt{cons}_a(\mathtt{cons}_b M) \to \mathtt{cons}_{ab} M$
2. $\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathbf{Y}\mathtt{cons}_L$ if $b \leq a$
3. $\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathbf{Y}\mathtt{cons}_R$ if $b \geq a$
4. $\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathtt{cons}_{b\backslash a} M$ if $a \sqsubseteq b$ and $a \neq b$
5. $\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathtt{cons}_{(a\sqcup b)\backslash a}(\mathtt{tail}_{(a\sqcup b)\backslash b} M)$ if $a \uparrow b, a \not\sqsubseteq b, b \not\sqsubseteq a, b \not\leq a$ and $a \not\leq b$
6. $\mathtt{head}_r(\mathtt{cons}_a M) \to \mathtt{true}$, if $a < r$;
7. $\mathtt{head}_r(\mathtt{cons}_a M) \to \mathtt{false}$, if $a > r$;
8. $\mathtt{pif}\,\mathtt{true}\; M\; N \to M$,
9. $\mathtt{pif}\,\mathtt{false}\; M\; N \to N$,
10. $\mathtt{pif}\; L\; (\mathtt{cons}_a M)\; (\mathtt{cons}_b N) \to$ if $a \sqcap b \neq \bot$
    $\mathtt{cons}_{a\sqcap b}(\mathtt{pif}\; L\; (\mathtt{cons}_{a\backslash a\sqcap b} M)(\mathtt{cons}_{b\backslash a\sqcap b} N))$,
11. $$\frac{N \to N'}{MN \to MN'}\ ,\ \text{if } M \text{ is } \mathtt{cons}_a, \mathtt{tail}_a, \mathtt{head}_r \text{ or } \mathtt{pif};$$

12.

$$\frac{L \to L'}{\texttt{pif}\ L \to \texttt{pif}\ L'}\,, \qquad \frac{M \to M'}{\texttt{pif}\ L\ M \to \texttt{pif}\ L\ M'}\,,$$

$$\frac{N \to N'}{\texttt{pif}\ L\ M\ N \to \texttt{pif}\ L\ M\ N'}\,.$$

The reader is referred to [23] where these rules are shown to be well-defined. It is important to notice that no maximal elements are produced as subscripts of $\texttt{cons}_a$ or $\texttt{tail}_a$. Despite the parallelism introduced by the `pif` rule to the language, it does not produce inconsistencies as the following lemma shows:

Lemma 5.2.1. *$M \to N$ implies $[\![M]\!]_\rho = [\![N]\!]_\rho$ for all terms M and N and any environment ρ.*

Proof. See [23] □

5.2.3 Denotational Semantics

We let $D_{\mathtt{I}} = \mathcal{I}$, i.e. $D_{\mathtt{I}}$ stand for the unit interval domain, and we extend the interpretation of $\mathcal{B}$ of $\mathcal{L}_{PCF}$ to $\mathcal{L}^I_{PCF}$ by defining the interpretation of the new constants as follows:

$$\begin{aligned}
\mathcal{B}[\![\texttt{cons}_a]\!] &= \mathsf{cons}_a,\\
\mathcal{B}[\![\texttt{tail}_a]\!] &= \mathsf{tail}_a,\\
\mathcal{B}[\![\texttt{head}_r]\!] &= \mathsf{head}_r,\\
\mathcal{B}[\![\texttt{pif}]\!] &= \mathsf{pif}.
\end{aligned}$$

The functions cons_a and tail_a were defined in Section 3.2.1. The function head_r is defined as:

$$\mathsf{head}_r(x) = \begin{cases} \mathsf{true}, & \text{if } x < r;\\ \mathsf{false}, & \text{if } r > x;\\ \bot, & \text{if } x \uparrow r; \end{cases}$$

and pif is defined as:

$$\mathsf{pif}(b)(x)(y) = \begin{cases} x, & \text{if } b = \texttt{true};\\ y, & \text{if } b = \texttt{false};\\ x \sqcap y, & \text{if } b = \bot. \end{cases}$$

5.2.4 Computational Adequacy

For programs of truth-value type and natural numbers type the partial map $\mathtt{Eval}$ is defined in the same way as for $\mathcal{L}_{PCF}$. It is well-defined as a consequence of Lemma 5.2.1, because no two different constants have the same interpretation.

The map $\mathtt{Eval}$ is extended to a multi-valued map on real programs M by

$$\mathtt{Eval}(M) = \{a \in \mathcal{I} \mid M \to^* \mathtt{cons}_a M' \text{ for some } M'\}.$$

Similar to PCF, in Real PCF the operational semantics and denotational semantics are related by computational adequacy. Here computational adequacy is proved in two stages called soundness and completeness.

Theorem 5.2.2 (Soundness). *For any real program M,*

$$\bigsqcup \mathtt{Eval}(M) \sqsubseteq [\![M]\!].$$

Proof. See [23] □

Theorem 5.2.3 (Completeness). *For any real program M,*

$$\bigsqcup \mathtt{Eval}(M) \sqsupseteq [\![M]\!].$$

Proof. The proof is by structural induction on the formation rules of terms [23]. □

Theorem 5.2.4 (Adequacy). *$\mathcal{L}^I_{PCF}$ is computationally adequate.*

Proof. Theorems 5.2.2 and 5.2.3 □

Part II

A Model for Sequential Computation on the Reals

Chapter 6

Description of the problem

In this chapter, the main aspects of the problem to be solved are described. In Section 6.1, we analize the problem that we solve and present our approach using a multi-valued construct. In Section 6.2, we discuss why it is important to consider the multi-valued fragment of the construction. In Section 6.3, we present the construction that Boehm and Cartwright considered in their operational approach to exact real-number computation, and we introduce the variation of this construction considered in this work.

6.1 The Problem

We develop a programming language for exact real-number computation. It will be recalled that in the introduction we set out the following requirements:

1. The language should be expressive for practical purposes, and ideally as expressive as the theory permits.

2. The datatype of real numbers should be abstract.

3. The language should have a sequential evaluation strategy.

Previous work on real-number computation has shown that it is possible to fulfil any combination of two of the above requirements, but that there are intrinsic difficulties in combining the three simultaneously. In this work, we seek a solution to this problem. Before proceeding to discuss the technical ingredients of our solution, we pause to discuss the above requirements and the difficulties of reconciling them which previous authors have found.

Expressivity. An expressive programming language is characterised by definability, e.g. a programming language in which we can define computable first order functions $f : \mathbb{R}^n \to \mathbb{R}$ is less expressive than a programming language in which we can define both computable first order functions and computable second order functions $f : (\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}$.

Data abstraction. This is an important requirement when constructing a programming language for exact real-number computation. With an abstract data type for real numbers, the programmer does not have access to representations within the programming language and can think of real numbers as abstract entities in the usual mathematical sense.

Sequentiality. Roughly speaking, this means that in order to evaluate an unevaluated program M, there should exist a subprogram of M which has to be evaluated first in order to evaluate M. In this work, we are interested in a sequential evaluation strategy. A complete presentation of sequentiality can be found in [3, Chapters 2.4 and 6.5].

Expressivity and data abstraction. As mentioned in the introduction, Di Gianantonio [15], Escardó [22], and Potts et al. [72] have introduced various extensions of the programming language PCF with a ground type for real numbers. In the three cases, the real number type is interpreted as a variation of the interval domain introduced by Scott [74]. In the presence of a certain parallel conditional [68], all computable first-order functions on the reals are definable in the languages [15, 23]. By further adding Plotkin's parallel existential quantifier [68], all computable functions of all orders become definable in the languages [15, 23, 28]. This shows that requirements 1 and 2 can be achieved simultaneously.

Data abstraction and sequentiality. Of course, if one sacrifices expressivity, the remaining two requirements can be easily achieved. For example, one can consider the extension of PCF introduced by Escardó [24] with all parallel constructions omitted. However Farjudian [29] has shown that all definable functions in this fragment of the language are piecewise linear, which rules out most of the functions which one would like to compute in practice.

Expressivity and sequentiality. Several implementations of exact real-number computations have been developed [10, 59, 72, 83]. In these works, real numbers are represented by:

- Signed digit representation [5, 88],
- continued fractions [7, 83, 51],
- functions mapping an integer specifying the precision required [10, 9, 59],
- golden-ratio notation [16],
- infinite sequences of nested intervals with rational end points [22],
- Möbius transformations [72],
- hybrid representations [75], among others.

Sequential implementations of fundamental computable first and second order functions have been developed using these representations, at the cost of sacrificing data abstraction.

Expressivity, data abstraction and sequentiality. The research discussed above 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. In our knowledge, there was no programming language satisfying these three requirements simultaneously. At the same time with this work, Farjudian [30] has developed a programming language, which he called SHRAD, with the above three requirements. In his work, he defines a sequential language in which all computable first order functions are definable. However extensionality[1] 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, intensional functions such as the rounding function, which is frequently used in practice, cannot be defined in SHRAD as we discuss in 6.2.

Based on ideas arising from constructive mathematics, Boehm and Cartwright [10] proposed a compelling operational solution to the problem using a multi-valued construction. However, they did not provide a semantics for their framework. Furthermore, it is not immediately clear what the corresponding treatment of the multi-valued construction, which is a relation but

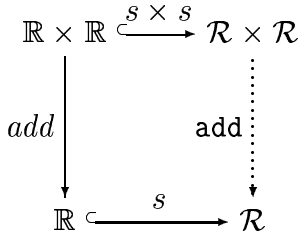
[1]Extensionality: For any computable function f and $d, e \in \mathcal{I}$, if $[\![d]\!] = [\![e]\!]$ then

$$[\![f(d)]\!] = [\![f(e)]\!]$$

not a function, would be. A partially successful attempt of solving this problem has been developed by Edalat, Potts and Sünderhauf [21], as discussed in the introduction.

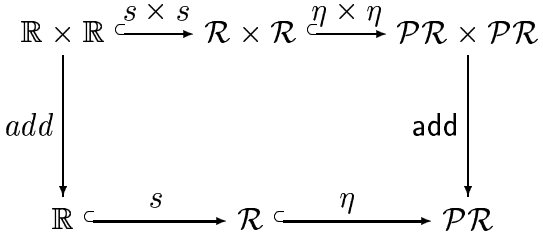
In this work, we establish the intrinsic difficulties of providing a semantics of Boehm and Cartwright's operational approach, and we show how it is possible to cope with the difficulties. Thus, we present a sufficiently expressive programming language (see Section 12.1) with sequential operational semantics and corresponding denotational semantics that articulate the data abstraction requirement.

Escardó, Hofmann and Streicher [27] have shown that the interval domain $\mathcal{R}$, which is the common interpretation of the data type of real numbers, cannot be used as a model for a programming language with a sequential operational semantics, e.g. if we consider a basic computable function such as addition, $add : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, there is no sequential computable down arrow $\mathsf{add} : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ that completes the following diagram:



Hence, the interval domain is ruled out as a possible model of the required programming language.

The model that we consider is a powerdomain construction of the interval domain. There are several powerdomain constructions in the literature [67, 78, 79, 33]. We present the conditions that a powerdomain construction should satisfy. Hence, we show which powerdomain constructions considered in the literature cannot be used as a model. It is shown that at least one powerdomain construction satisfies these conditions, which is known as the Hoare powerdomain construction. We show that if we consider a computable function for addition $add : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, there is a sequentially computable down arrow function $\mathsf{add} : \mathcal{PR} \times \mathcal{PR} \to \mathcal{PR}$ that makes the following diagram commute:



when $\mathcal{PR}$ is considered as the Hoare powerdomain construction. Hence we are avoiding parallel constructions by not considering that the "middle square" exists.

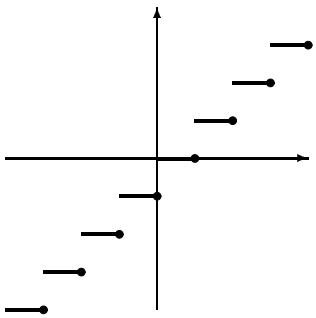
6.2 Relevance of the multi-valued construction

The programming language which we present in Chapter 9 has a multi-valued constructor among its constants. In order to give a full interpretation of the programming language, we should give an interpretation of this construction in terms of multi-valued functions. It is very important to consider the semantics of multi-valued constructions for at least four reasons.

- It is well-known that there are several practical problems, like the determination of zeros of a polynomial, which only admit multi-valued computable solutions but no single-valued ones. Here multi-valued functions are interesting in their own right. We present a program for root finding in Section 10.3

- It is recalled that in Section 4.5 we show that in exact real number computation, every computable function is continuous. Hence, easily definable functions in Mathematics such as the upper integer bound

$$f(x) = \lceil x \rceil$$

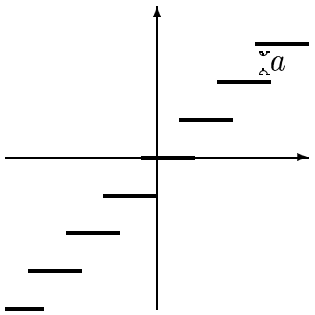


are not computable. However, if we allow the upper bound function to be multi-valued, it becomes computable.

Definition 6.2.1 (computable upper integer bound function). *For every $a \in \mathbb{Q}$ and every $x \in \mathbb{R}$ such that $(\lceil x \rceil - 1) < a < \lceil x \rceil$, the continuous upper integer bound function is defined as:*

$$\texttt{upper\_bound}_a : \mathbb{R} \to \mathcal{P}\mathbb{N}$$

$$\texttt{upper\_bound}_a(x) = \begin{cases} \lceil x \rceil, & \text{if } x < \lceil x \rceil; \\ \lceil x \rceil + 1 & \text{if } a < x; \end{cases}$$



Notice that if $x \in (a, \lceil x \rceil)$ then $\texttt{upper\_bound}(x) = \{\lceil x \rceil, \lceil x \rceil + 1\}$. We present a continuous extension of the computable upper integer bound function in Section 8.6.

- Some functions (such as average) admit single-valued implementations only at the cost of allowing non-sequential constructions. However in Chapter 8 we present an implementation of average which is single-valued at total inputs, but multi-valued at partial inputs and this is unavoidable otherwise by the results of Escardó, Hofmann and Streicher [27] mentioned above, one would be able to define parallel operators from the average function.

- If we want to compute a power series $\sum_{n=1}^{\infty} c_n z^n$, it is known that it has an associated circle of converges, such that $\sum_{n=1}^{\infty} c_n z^n$ converges if z is in the interior of the circle. For example, the power series

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{(x-1)^n}{n} \tag{6.1}$$

calculates the natural logarithm of x (denoted by $\mathsf{ln}(x)$) with an interval of convergence $0 < x \leq 2$. Suppose that we are interested in calculating $\mathsf{ln}(x)$ with an interval of convergence $0 < x \leq 3$ ($\mathsf{ln}(x)$ is an increasing continuous function); hence we can use 6.1 together with

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{(x-1.5)^n}{n} \tag{6.2}$$

which is known to converge in the interval $1.5 < x \leq 3$. Using 6.1 and 6.2, we have two definitions to choose from on the interval $1.5 \leq$

$x \leq 2$; hence we can use the multi-valued construction to present the required definition. This particular example can be applied to other power series defined in a similar way.

Our study of the semantics in the general multi-valued case is built on this basis.

6.3 Boehm and Cartwright approach

The essential problem of constructing an expressive programming language with an abstract data type for real numbers and sequential operational semantics has already been presented in the Introduction.

Boehm and Cartwright observed that an expressive programming language with an abstract data type for real numbers must contain either a parallel construction or a redundant test construction. Their idea, using a substitute for the trichotomy law in constructive mathematics, has been discussed in the introduction of the thesis.

Boehm and Cartwright defined a redundant test construction which they called a quasi-relational comparison operator $<_\epsilon : \mathbb{R} \times \mathbb{R} \to \mathbb{T}$, defined as:

1. $<_\epsilon (x, y)$ evaluates to true or to false for every pair in $\mathbb{R} \times \mathbb{R}$,
2. if $<_\epsilon (x, y)$ evaluates to true then $x < y - \epsilon$, and
3. if $<_\epsilon (x, y)$ evaluates to false then $x \geq y$.

where ϵ is a rational number.

We work with an equivalent variation of the quasi-relational operation which we call $\mathsf{rtest}_{a,b} : \mathbb{R} \to \mathbb{T}$ for $a < b$ rational, such that

1. $\mathsf{rtest}_{a,b}(x)$ evaluates to true or to false for every element in $\mathbb{R}$,
2. $\mathsf{rtest}_{a,b}(x)$ may evaluate to true iff $x < b$, and
3. $\mathsf{rtest}_{a,b}(x)$ 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. [56]), and is thus determined by the operational semantics.

Boehm and Cartwright did not provide an operational semantics or a denotational semantics for a language using their quasi-relational comparison operator. They focused mainly on the efficiency of some algorithms using their approach. However, it is expedient to investigate whether their observation is in fact admissible and if so, what kind of denotational representations can be used in order to make the construction computable. We take the view that the denotational value of $\mathsf{rtest}_{a,b}(x)$ lives in a suitable powerdomain of the booleans and study the possible denotational representations for $\mathsf{rtest}_{a,b}$ in the next chapter.

The principal advantage of $\mathsf{rtest}_{a,b}$ over the parallel conditional is that it does not require dovetailed evaluation, making it much easier to implement efficiently. On the other hand, it is less abstract that the parallel conditional because its behaviour depends on the representation of its arguments.

Chapter 7

Mathematical Interpretation of the Multi-valued Construction

In order to continue the development of Chapter 6, we assume that for every domain D, we have a functorial powerdomain construction $\mathcal{P}$ together with a natural embedding

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

and a continuous formal-union operation

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

When no confusion arises, we simply write η. Then the definition of the function $\mathsf{rtest}_{a,b} \colon \mathbb{R} \to \mathcal{P}\mathbb{T}$, where $a < b$ are real numbers, can be formulated as

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

Because in our language there will be computations on the real numbers which 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 3.1. Similarly, as usual, we enlarge the set $\mathbb{T}$ 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:

$$\begin{array}{ccc} \mathbb{R} & \xrightarrow{\mathsf{rtest}_{a,b}} & \mathcal{P}\mathbb{T} \\ \downarrow & & \downarrow \\ \mathcal{R}_\perp & \xrightarrow{\mathsf{rtest}_{a,b}} & \mathcal{P}\mathbb{T}_\perp. \end{array} \tag{7.1}$$

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 there is at least one continuous extension. The following formulates necessary conditions for this.

Lemma 7.0.1. *If* $\mathsf{rtest}_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ *is a continuous extension of the function* $\mathsf{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$*, 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 $\mathcal{R}_\perp$ is endowed with its Scott topology, its composition with the function $\mathsf{rtest}_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ is continuous, 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 the continuity of $r : \mathbb{R} \to \mathcal{P}\mathbb{T}_\perp$. Because $r(a) = t \in V$, it follows 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 the 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 as Figure 7.2 shows. For the Smyth powerdomain, the same sets are obtained by the embedding, formal union is given by actual union, and hence the

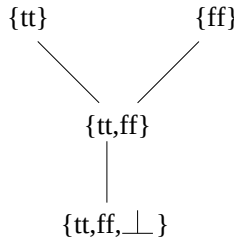


Figure 7.1: Smyth Power Domain on $\mathbb{T}_\perp$.

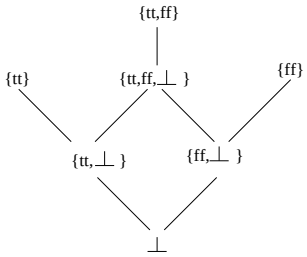


Figure 7.2: Plotkin Power Domain on $\mathbb{T}_\perp$.

inequalities do not hold because the order is given by reverse inclusion as Figure 7.1 shows. We formulate this conclusion for emphasis:

Corollary 7.0.2. *When $\mathcal{P}$ is the Smyth or Plotkin powerdomain, there is no continuous extension $\mathcal{R}_\perp \to \mathcal{P}\mathbb{T}_\perp$ of the function $\mathsf{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$.*

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:

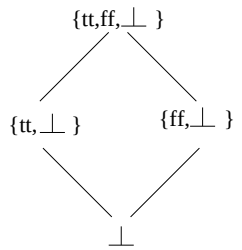


Figure 7.3: Hoare Powerdomain on $\mathbb{T}_\perp$.

Proposition 7.0.3. *There is a continuous extension* $\mathsf{rtest}^H_{a,b}\colon \mathcal{R}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ *of the function* $\mathsf{rtest}_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$.

Proof. The functions $f, g\colon \mathcal{R}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ 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, their join

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

is well-defined and continuous because $\mathcal{P}^H D$ is always a complete lattice. It is easy to show that this function has the required extension property:

- If $x \subseteq (a, b)$ then $(f \sqcup g)(x) = (f(x)) \sqcup (g(x)) = \eta(\mathsf{true}) \sqcup \eta(\mathsf{false}) = \eta(\mathsf{true}) \uplus \eta(\mathsf{false}) = \{\mathsf{true}, \mathsf{false}\}$.
- If $x \subseteq (-\infty, b)$ but $x \not\subseteq (a, b)$, then $(f \sqcup g)(x) = (f(x)) \sqcup (g(x)) = \eta(\mathsf{true}) \sqcup \eta(\perp) = \eta(\mathsf{true}) \uplus \eta(\perp) = \{\mathsf{true}, \perp\}$.
- if $x \subseteq (a, \infty)$ but $x \not\subseteq (a, b)$, then $(f \sqcup g)(x) = (f(x)) \sqcup (g(x)) = \eta(\perp) \sqcup \eta(\mathsf{false}) = \eta(\perp) \uplus \eta(\mathsf{false}) = \{\mathsf{false}, \perp\}$.
- if x does not satisfy any of the above requirements, then $(f \sqcup g)(x) = (f(x)) \sqcup (g(x)) = \eta(\perp) \sqcup \eta(\perp) = \eta(\perp) \uplus \eta(\perp) = \perp$.

□

Regarding the Sandwich powerdomain (see Section 2.5.5) and the Mixed powerdomain (see Section 2.5.6), it is immediately apparent by the given order that they do not satisfy the conditions to make $\mathsf{rtest}_{a,b}$ continuous (see Figures 7.4 and 7.5). In both cases the sets

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

are incomparable as in the case of the Plotkin powerdomain. 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.

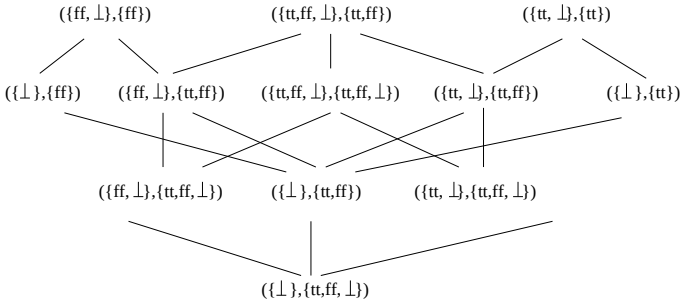


Figure 7.4: Sandwich Powerdomain on $\mathbb{T}_\perp$.

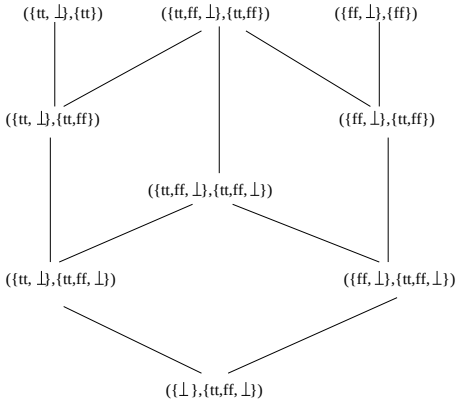


Figure 7.5: Mixed Powerdomain on $\mathbb{T}_\perp$.

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, where the latter is established by operational methods. A related approach is considered in Chapter 10.

Remark 7.0.4. One could consider a natural variation $\mathrm{rtest}'_{a,b} : \mathbb{R} \to \mathcal{P}\mathbb{T}$ of the $\mathsf{rtest}_{a,b}$ construction, defined by

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

It can be immediately notice the difference with $\mathsf{rtest}_{a,b}(x)$ which returns $\eta(\mathsf{true}) \uplus \eta(\mathsf{false})$ if $x \in (a,b)$; $\eta(\mathsf{true})$ if $x \in (-\infty, a]$ and $\eta(\mathsf{false})$ if $[b,\infty)$.

With a proof similar to that of Lemma 7.0.1, 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 retain the construction as originally proposed by Boehm and Cartwright.

Chapter 8

Recursive Definitions with the Multi-valued Construction

In this chapter, we define the continuous extended functions of basic arithmetic operations using the $\mathsf{rtest}_{a,b} : \mathcal{I} \to \mathcal{P}^H\mathbb{T}_\perp$ construction and elementary continuous functions. These arithmetic operations are defined recursively, hence we consider the continuous extension $\overline{\mathsf{rtest}}_{a,b} : \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathbb{T}_\perp$ of the continuous function $\mathsf{rtest}_{a,b} : \mathcal{I} \to \mathcal{P}^H\mathbb{T}_\perp$. It will be recalled that in Section 2.5.4 we set out this extension by:

$$\overline{\mathsf{rtest}}_{a,b}(X) = \mathsf{cl}\left(\bigcup_{x \in X} \mathsf{rtest}_{a,b}(x)\right).$$

In our definitions, we require continuous extensions of the functions cons_a, tail_a (see Section 3.2.1) and $\mathcal{B}[\![\mathtt{if}]\!]$ (see Section 5.1). It will be recalled again that in Section 2.5.4 we set out these extensions by:

$$\begin{array}{rccl} \widehat{\mathsf{cons}}_a : & \mathcal{P}^H\mathcal{I} & \to & \mathcal{P}^H\mathcal{I} \\ & X & \mapsto & \mathsf{cl}\{\mathsf{cons}_a(x) \mid x \in X\} \end{array}$$

$$\begin{array}{rccl} \widehat{\mathsf{tail}}_a : & \mathcal{P}^H\mathcal{I} & \to & \mathcal{P}^H\mathcal{I} \\ & X & \mapsto & \mathsf{cl}\{\mathsf{tail}_a(x) \mid x \in X\} \end{array}$$

$$\mathsf{if} : \mathcal{P}^H\mathbb{T}_\perp \times \mathcal{P}^H\sigma \times \mathcal{P}^H\sigma \to \mathcal{P}^H\sigma$$

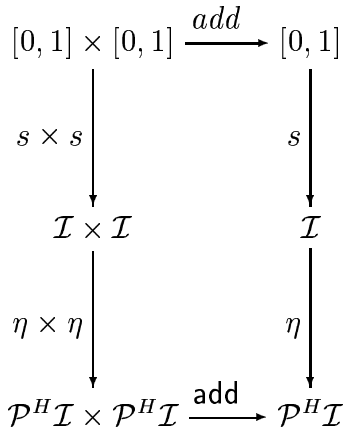
$$\mathsf{if}(B, X, Y) = \begin{cases} X, & \text{if } B = \eta(\mathsf{true}); \\ Y, & \text{if } B = \eta(\mathsf{false}); \\ X \uplus Y, & \text{if } B = \eta(\mathsf{true}) \uplus \eta(\mathsf{false}); \\ \{\bot\}, & B = \{\bot\}. \end{cases}$$

The aim of this section is the following: given a continuous function $f : [0,1] \to [0,1]$, we want to find a continuous extension of f such that the following diagram commutes:

$$\begin{array}{ccc} [0,1] & \xrightarrow{\;f\;} & [0,1] \\ \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle s} \\ \mathcal{I} & & \mathcal{I} \\ \downarrow{\scriptstyle \eta} & & \downarrow{\scriptstyle \eta} \\ \mathcal{P}^H\mathcal{I} & \xrightarrow{\;\mathsf{f}\;} & \mathcal{P}^H\mathcal{I} \end{array}$$

where s is the continuous embedding map defined as $x \mapsto \{x\}$, see Chapter 3, and η is the continuous inclusion map defined as $x \mapsto {\downarrow}x$, see Section 2.5. The main result of this chapter follows from the commutativity of the above diagram. Escardó, Hofmann and Streicher [27] have shown that for elementary functions such as addition, there is no sequential map $\mathsf{add} : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ such that $add \circ s = (s \times s) \circ \mathsf{add}$. More precisely, if there is a function for addition $add : [0,1] \times [0,1] \to [0,1]$, any continuous extension of add in the interval domain $\mathsf{add} : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ together with some manifestly sequential unary functions defines weak parallel-or. However, we show that there is a sequential continuous extension $\mathsf{add} : \mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I}$ of add, for which

the following diagram is commutative:



Thus, we overcome the problem by allowing our definition to be multi-valued at partial inputs. Our main result shows that the single-valued output 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.

Besides the average function, we present sequential extension for the absolute value function, the complement function, the multiplication function and the division function.

8.1 Complement Function

We derive a recursive definition of a continuous extension of the complement map

$$comp : [0,1] \to [0,1]$$

$$comp(x) = 1 - x.$$

The extended map is given by $\mathsf{comp} : \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I}$

$\mathsf{comp}(C)$= if $\overline{\mathsf{rtest}}_{1/3,2/3}(C)$
then
if $\overline{\mathsf{rtest}}_{1/3,1/2}(C)$
then $\widehat{\mathsf{cons}}_R(\mathsf{comp}(\widehat{\mathsf{tail}}_L(C)))$
else $\widehat{\mathsf{cons}}_C(\mathsf{comp}(\widehat{\mathsf{tail}}_C(C)))$
else

$$\begin{array}{l}\mathsf{if}\ \overline{\mathsf{rtest}}_{1/2,2/3}(C)\\ \quad \mathsf{then}\ \widehat{\mathsf{cons}}_C(\mathsf{comp}(\widehat{\mathsf{tail}}_C(C)))\\ \quad \mathsf{else}\ \widehat{\mathsf{cons}}_L(\mathsf{comp}(\widehat{\mathsf{tail}}_R(C)))\end{array}$$

For maximal partial numbers, we show that it indeed coincides with the original map. We let $L = [0, 1/2], R = [1/2, 1]$ and $C = [1/3, 2/3]$; then we have the following definitions of the cons and tail maps according to Section 3.2.1:

$$\begin{array}{ll}\mathsf{cons}_L(x) = x/2, & \mathsf{tail}_L(x) = \mathsf{min}\{2x, 1\},\\ \mathsf{cons}_R(x) = (x+1)/2, & \mathsf{tail}_R(x) = \mathsf{max}\{0, 2x-1\},\\ \mathsf{cons}_C(x) = (x+1)/3, & \mathsf{tail}_C(x) = \mathsf{min}\{\mathsf{max}\{0, 3x-1\}, 1\}.\end{array}$$

According to the definition of the complement function, the next observations apply:

(i) if $x \subseteq L$ then $1 - x \subseteq R$,

(ii) if $x \subseteq R$ then $1 - x \subseteq L$,

(iii) if $x \subseteq C$ then $1 - x \subseteq C$.

Theorem 8.1.1. *For all $x \in [0, 1]$ it holds that* $\mathsf{comp}(\eta(x)) = \eta(comp(x))$.

Proof. Take $D = (\mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I})$. Define $\Phi : D \to D$ by, for all $F \in D$,

$$\begin{array}{l}\Phi(F)(C) = \mathsf{if}\ \overline{\mathsf{rtest}}_{1/3,2/3}(C)\\ \qquad \mathsf{then}\\ \qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{1/3,1/2}(C)\\ \qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_R(F(\widehat{\mathsf{tail}}_L(C)))\\ \qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_C(F(\widehat{\mathsf{tail}}_C(C)))\\ \qquad \mathsf{else}\\ \qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{1/2,2/3}(C)\\ \qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_C(F(\widehat{\mathsf{tail}}_C(C)))\\ \qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_L(F(\widehat{\mathsf{tail}}_R(C)))\end{array}$$

so that $\mathsf{comp} = \bigsqcup_n \mathsf{comp}_n$ where $\mathsf{comp}_n = \Phi^n(\bot)$. To conclude we use Lemma 8.1.2. □

Lemma 8.1.2. *For any $x \in [0, 1]$, the following conditions hold:*

1. $\mathsf{comp}_n(\eta(x))$ *is of the form* $\downarrow F_n$ *for* $F_n \subseteq \mathcal{I}$ *finite,*
2. $\kappa_y \leq 2^{-n+1}$ *for each* $y \in F_n$,

3. $F_n \sqsubseteq \eta(comp(x))$.

Proof. The proof is by induction on n.

$n = 0$. We know that $\mathsf{comp}_0(\eta(x)) = \{\bot\} = \downarrow\{\bot\}$ for any $x \in [0,1]$. Take $y \in F_n = \{\bot\}$, so $\kappa_y = 1 < 2^{-n+1} = 2$, and $\{\bot\} \sqsubseteq \eta(comp(x))$ for all $x \in [0,1]$.

Assume that it holds for n. We prove that it holds for $n+1$. We proceed according to the position of x.

a) If $x \in L \cup C$, we have three cases:

1. if $x \in (L)^\circ \setminus (C)^\circ$

   In this case:

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

   by the inductive hypothesis, $\mathsf{comp}_n(\eta(t))$ is of the form $\downarrow F_n$ for F_n finite and $t = \mathsf{tail}_L(x)$. Take $F_{n+1} = \bigcup_{w \in F_n} \mathsf{cons}_R(w)$ then $\mathsf{comp}_{n+1}(\eta(x))$ is of the form $\downarrow\{\bigcup_{w \in F_n} \mathsf{cons}_R(w)\}$. F_{n+1} is finite because F_n is.

   Let $y \in F_{n+1}$. We must show that $\kappa_y \leq 2^{-n}$. We know that $y = \mathsf{cons}_R(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq 2^{-n+1}$. If $y = \mathsf{cons}_R(t) = \frac{t+1}{2}$ then:

$$\overline{t} - \underline{t} \leq 2^{-n+1}$$

$$\frac{\overline{t}}{2} - \frac{\underline{t}}{2} \leq \frac{1}{2}(2^{-n+1}) = 2^{-n}$$

   hence $\kappa_y \leq 2^{-n}$.

   Let $y \in F_{n+1}$. We must show that $\{y\} \sqsubseteq \eta(comp(x))$. As $y \in F_{n+1} = \widehat{\mathsf{cons}}_R(F_n)$, so $y = \mathsf{cons}_R(s)$ for some $s \in F_n$. By the inductive hypothesis for each $s \in F_n$, $\{s\} \sqsubseteq \eta(comp(\mathsf{tail}_L(x)))$, hence:

$$\begin{aligned}\{y\} = \{\mathsf{cons}_R(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_R(\eta(comp(\mathsf{tail}_L(x))))\\ \text{(definition of } \mathsf{tail}_L) &= \widehat{\mathsf{cons}}_R(\eta(comp(2x)))\\ \text{(definition of } comp) &= \widehat{\mathsf{cons}}_R(\eta(1-2x))\\ \text{(Diagram 3.1)} &= \eta(\mathsf{cons}_R(1-2x))\\ \text{(definition of } \mathsf{cons}_R) &= \eta(1-x) = \eta(comp(x)),\end{aligned}$$

   as we wanted.

2. if $x \in (C)^\circ \setminus (L)^\circ$, in this case:

$$\begin{aligned}\mathsf{comp}_{n+1}(\eta(x)) &= \widehat{\mathsf{cons}}_C(\mathsf{comp}_n(\widehat{\mathsf{tail}}_C(\eta(x)))) \\ &\quad \uplus \widehat{\mathsf{cons}}_L(\mathsf{comp}_n(\widehat{\mathsf{tail}}_R(\eta(x)))) \\ &= \widehat{\mathsf{cons}}_C(\mathsf{comp}_n(\eta(\mathsf{tail}_C(x)))) \\ &\quad \uplus \widehat{\mathsf{cons}}_L(\mathsf{comp}_n(\eta(\mathsf{tail}_R(x)))),\end{aligned}$$

by the inductive hypothesis, $\mathsf{comp}_n(\eta(t))$ is of the form $\downarrow F_n$ for F_n finite. Take $F_{n+1} = \mathsf{cons}_C(F_n) \uplus \mathsf{cons}_L(F_n)$ then $\mathsf{comp}_{n+1}(\eta(x))$ is of the form $\downarrow\{\mathsf{cons}_C(F_n) \uplus \mathsf{cons}_L(F_n)\}$. F_{n+1} is finite because F_n is.

Let $y \in F_{n+1}$. We must show that $\kappa_y \leq 2^{-n}$. We know that either $y = \mathsf{cons}_C(t)$ or $y = \mathsf{cons}_L(t)$ for some $t \in F_n$. By inductive hypothesis $\kappa_t \leq 2^{-n+1}$. There are two cases:

if $y = \mathsf{cons}_C(t) = \frac{t+1}{3}$ then:

$$\overline{t} - \underline{t} \leq 2^{-n+1},$$

$$\frac{\overline{t}}{3} - \frac{\underline{t}}{3} \leq \frac{1}{3}(2^{-n+1}) \leq 2^{-n},$$

hence $\kappa_y \leq 2^{-n}$.

If $y = \mathsf{cons}_L(t) = \frac{t}{2}$ then:

$$\overline{t} - \underline{t} \leq 2^{-n+1},$$

$$\frac{\overline{t}}{2} - \frac{\underline{t}}{2} \leq \frac{1}{2}(2^{-n+1}) = 2^{-n},$$

hence $\kappa_y \leq 2^{-n}$.

Let $y \in F_{n+1}$. We must show that $\{y\} \sqsubseteq \eta(comp(x))$. As $y \in F_{n+1} = \widehat{\mathsf{cons}}_C(F_n) \uplus \widehat{\mathsf{cons}}_L(F_n)$ so $y = \mathsf{cons}_C(s)$ or $y = \mathsf{cons}_L(s)$ for some $s \in F_n$. By inductive hypothesis for each $s \in F_n$, either $\{s\} \sqsubseteq \eta(comp(\mathsf{tail}_C(x)))$ or $\{s\} \sqsubseteq \eta(comp(\mathsf{tail}_R(x)))$, hence:

$$\begin{aligned}\{y\} = \{\mathsf{cons}_C(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_C(\eta(comp(\mathsf{tail}_C(x)))) \\ (\text{definition of } \mathsf{tail}_C) &= \widehat{\mathsf{cons}}_C(\eta(comp(3x-1))) \\ (\text{definition of } comp) &= \widehat{\mathsf{cons}}_C(\eta(1-(3x-1))) \\ (\text{Diagram 3.1}) &= \eta(\mathsf{cons}_C(3x)) \\ (\text{definition of } \mathsf{cons}_C) &= \eta(1-x) = \eta(comp(x)),\end{aligned}$$

or

$$\begin{aligned}
\{y\} = \{\mathsf{cons}_L(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_L(\eta(comp(\mathsf{tail}_R(x)))) \\
(\text{definition of } \mathsf{tail}_R) &= \widehat{\mathsf{cons}}_L(\eta(comp(2x-1))) \\
(\text{definition of } comp) &= \widehat{\mathsf{cons}}_L(\eta(1-(2x-1))) \\
(\text{Diagram 3.1}) &= \eta(\mathsf{cons}_L(2x)) \\
(\text{definition of } \mathsf{cons}_R) &= \eta(1-x) = \eta(comp(x)),
\end{aligned}$$

as we wanted.

3. if $x \in (L) \cap (C)$

   The proof is similar to 2, using cons_R instead of cons_L

b) If $x \in C \cup R$, again we have three cases:

1. if $x \in R \setminus C$

   The proof is similar to a.1, Using cons_L instead of cons_R.

2. if $x \in C \setminus R$

   The proof is identical to a.3

3. if $x \in C \cap R$

   The proof is identical to a.2

c) If $x \in C \cap L \cap R$, in this case:

$$\begin{aligned}
\mathsf{comp}_{n+1}(\eta(x)) &= \widehat{\mathsf{cons}}_C(\mathsf{comp}_n(\widehat{\mathsf{tail}}_C(\eta(x)))) \\
(\text{Diagram 3.2}) &= \widehat{\mathsf{cons}}_C(\mathsf{comp}_n(\eta(\mathsf{tail}_C(x)))),
\end{aligned}$$

by the inductive hypothesis, $\mathsf{comp}_n(\eta(t))$ is of the form $\downarrow F_n$ for F_n finite. Take $F_{n+1} = \mathsf{cons}_C(F_n)$; then $\mathsf{comp}_{n+1}(\eta(x))$ is of the form $\downarrow \mathsf{cons}_C(F_n)$. F_{n+1} is finite because F_n is.

Let $y \in F_{n+1}$. We must show that $\kappa_y \leq 2^{-n}$. We know that $y = \mathsf{cons}_C(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq 2^{-n+1}$. With this we have: $y = \mathsf{cons}_C(t) = \frac{t+1}{3}$. Then:

$$\overline{t} - \underline{t} \leq 2^{-n+1},$$

$$\frac{\overline{t}}{3} - \frac{\underline{t}}{3} \leq \frac{1}{3}(2^{-n+1}) \leq 2^{-n},$$

hence $\kappa_y \leq 2^{-n}$.

Let $y \in F_{n+1}$. We must show that $\{y\} \sqsubseteq \eta(comp(x))$. As $y \in F_{n+1} = \widehat{\mathsf{cons}}_C(F_n)$ so $y = \mathsf{cons}_C(s)$ for some $s \in F_n$. By the inductive hypothesis for each $s \in F_n$, $\{s\} \sqsubseteq \eta(comp(\mathsf{tail}_C(x)))$, hence:

$$\begin{aligned}
\{y\} = \{\widehat{\mathsf{cons}}_C(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_C(\eta(comp(\mathsf{tail}_C(x)))) \\
\text{(definition of } \mathsf{tail}_C) &= \widehat{\mathsf{cons}}_C(\eta(comp(3x-1))) \\
\text{(definition of } comp) &= \widehat{\mathsf{cons}}_C(\eta(1-(3x-1))) \\
\text{(Diagram 3.1)} &= \eta(\mathsf{cons}_C(3x)) \\
\text{(definition of } \mathsf{cons}_C) &= \eta(1-x) = \eta(comp(x)),
\end{aligned}$$

as we wanted. □

8.2 Absolute Value Function

We derive a recursive definition of a continuous extension of the absolute value function

$$abs : [-1/2, 1/2] \to [0, 1/2]$$

$$abs(x) = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{otherwise,} \end{cases}$$

In this case instead of considering the interval $[0,1]$, we work with the interval $[-1/2, 1/2]$. As explained in Section 3.1 this consideration does not affect the theory. In this way we can observe the results when a negative input is given. The extended map is given by $\mathsf{abs} : \mathcal{P}^H\mathcal{I}[-1/2,1/2] \to \mathcal{P}^H\mathcal{I}[0,1/2]$

$\mathsf{abs}(C)$= if $\overline{\mathsf{rtest}}_{-1/3,1/3}(C)$
then
if $\overline{\mathsf{rtest}}_{-1/3,0}(C)$
then $\widehat{\mathsf{cons}}_R(\mathsf{comp}(\widehat{\mathsf{tail}}_L(C)))$
else $\widehat{\mathsf{cons}}_C(\mathsf{abs}(\widehat{\mathsf{tail}}_C(C)))$
else
if $\overline{\mathsf{rtest}}_{0,1/3}(C)$
then $\widehat{\mathsf{cons}}_C(\mathsf{abs}(\widehat{\mathsf{tail}}_C(C)))$
else $\widehat{\mathsf{cons}}_R(\mathsf{abs}(\widehat{\mathsf{tail}}_R(C)))$

This map coincides with the maximal partial numbers of the original map. Let $L = [-1/2, 0], C = [-1/3, 1/3]$ and $R = [0, 1/2]$, hence cons and tail are

defined as:

$$\begin{aligned}
\mathsf{cons}_L(x) &= \frac{1}{2}x - \frac{1}{4}, & \mathsf{tail}_L(x) &= \mathsf{min}\left\{2x + \frac{1}{2}, \frac{1}{2}\right\},\\
\mathsf{cons}_R(x) &= \frac{1}{2}x + \frac{1}{4}, & \mathsf{tail}_R(x) &= \mathsf{max}\left\{-\frac{1}{2}, 2x - \frac{1}{2}\right\},\\
\mathsf{cons}_C(x) &= \frac{2}{3}x, & \mathsf{tail}_C(x) &= \mathsf{max}\left\{-\frac{1}{2}, \mathsf{min}\left\{\frac{3}{2}x, \frac{1}{2}\right\}\right\}.
\end{aligned}$$

The recursive definition for abs presented above is derived from a case analysis of the possible values of x as the following shows: $\forall y \in [-1/2, 1/2]$,

$$abs(\mathsf{cons_L}(y)) = abs\left(\frac{1}{2}y - \frac{1}{4}\right) = -\left(\frac{1}{2}y - \frac{1}{4}\right) = \frac{1}{2}(-y) + \frac{1}{4} = \mathsf{cons_R}(-y).$$

$$abs(\mathsf{cons_C}(y)) = abs\left(\frac{2y}{3}\right) = \frac{abs(2y)}{3} = \frac{2}{3}abs\,(y) = \mathsf{cons_C}(abs(y)).$$

$$abs(\mathsf{cons_R}(y)) = \mathsf{cons_R}(y).$$

Theorem 8.2.1. $\mathsf{abs}(\eta(x)) = \eta(abs(x))$ *for all* $x \in [-1/2, 1/2]$.

Proof. Take $D = (\mathcal{P}^H[-1/2, 1/2] \to \mathcal{P}^H[0, 1/2])$. Define $\Phi : D \to D$ by, for all $F \in D$,

$$\begin{array}{l}
\Phi(F)(C) = \mathsf{if}\ \overline{\mathsf{rtest}}_{-1/3,1/3}(C)\\
\qquad\qquad \mathsf{then}\\
\qquad\qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{-1/3,0}(C)\\
\qquad\qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_R(\mathsf{comp}(\widehat{\mathsf{tail}}_L(C)))\\
\qquad\qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_C(F(\widehat{\mathsf{tail}}_C(C)))\\
\qquad\qquad \mathsf{else}\\
\qquad\qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{0,1/3}(C)\\
\qquad\qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_C(F(\widehat{\mathsf{tail}}_C(C)))\\
\qquad\qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_R(\mathsf{comp}(\widehat{\mathsf{tail}}_R(C)))
\end{array}$$

so that $\mathsf{abs} = \bigsqcup_n \mathsf{abs}_n$ where $\mathsf{abs}_n = \Phi^n(\bot)$. To conclude we use Lemma 8.2.2. □

Lemma 8.2.2. *For any* $x \in [-1/2, 1/2]$, *the following conditions hold:*

1. $\mathsf{abs}_n(\eta(x))$ *is of the form* $\downarrow F_n$ *for* $F_n \subseteq \mathcal{I}[-1/2, 1/2]$ *finite,*

2. $\kappa_y \leq 2^{-n+1}$ *for each* $y \in F_n$,

3. $F_n \sqsubseteq \eta(abs(x))$.

Proof. The proof is by induction on n, and is similar to the proof of the complement function. The difference resides in the cases where x appears. □

8.3 Average Function

We now derive a recursive definition of a continuous extension of the average operation

$$\oplus : [0,1] \times [0,1] \to [0,1]$$

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

The extended map is given by $\mathsf{Average} : \mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I}$

$$\begin{array}{l}
\mathsf{Average}(X,Y) = \ \mathsf{if}\ \overline{\mathsf{rtest}}_{1/4,3/4}(X) \\
\qquad\qquad\qquad\qquad \mathsf{then} \\
\qquad\qquad\qquad\qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{1/4,3/4}(Y) \\
\qquad\qquad\qquad\qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_L(\mathsf{Average}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_L(Y))) \\
\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_C(\mathsf{Average}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_R(Y))) \\
\qquad\qquad\qquad\qquad \mathsf{else} \\
\qquad\qquad\qquad\qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{1/4,3/4}(Y) \\
\qquad\qquad\qquad\qquad\qquad \mathsf{then}\ \widehat{\mathsf{cons}}_C(\mathsf{Average}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_L(Y))) \\
\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_R(\mathsf{Average}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_R(Y)))
\end{array}$$

where $L = [0, 3/4], C = [1/8, 7/8], R = [1/4, 1]$.

For maximal partial elements we show that it indeed coincides with the original map.

Theorem 8.3.1. $\mathsf{Average}(\eta(x)), \eta(y)) = \eta\left(\frac{x+y}{2}\right)$ *for all* $x, y \in [0,1]$.

Proof. We take $D = (\mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I})$. Define $\Phi : D \to D$ by, for all $\mathsf{F} \in D$,

$$\begin{array}{l}
\Phi(\mathsf{F})(X,Y) = \ \mathsf{if}\ \overline{\mathsf{rtest}}_{1/4,3/4}(X) \\
\qquad\qquad\qquad\qquad \mathsf{then} \\
\qquad\qquad\qquad\qquad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{1/4,3/4}(Y)
\end{array}$$

$$
\begin{array}{l}
\qquad\qquad\quad \mathsf{then}\ \ \widehat{\mathsf{cons}}_L(\mathsf{F}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_L(Y))) \\
\qquad\qquad\quad \mathsf{else}\ \ \widehat{\mathsf{cons}}_C(\mathsf{F}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_R(Y))) \\
\quad \mathsf{else} \\
\qquad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{1/4,3/4}(Y) \\
\qquad\qquad\quad \mathsf{then}\ \ \widehat{\mathsf{cons}}_C(\mathsf{F}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_L(Y))) \\
\qquad\qquad\quad \mathsf{else}\ \ \widehat{\mathsf{cons}}_R(\mathsf{F}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_R(Y)))\,,
\end{array}
$$

so that $\mathsf{Average} = \bigsqcup_n \mathsf{Average}_n$, where $\mathsf{Average}_n = \Phi^n(\bot)$. To conclude, we use Lemma 8.3.2. □

Lemma 8.3.2. *For any $x, y \in [0,1]$, 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 $\{\bot\} \sqsubseteq \eta(x \oplus y)$ for all $x, y \in [0,1]$.

2. Assume that it is true for n. We prove that it is true for $n+1$. We proceed according the position of x and y.

a) if $x \in [0, 1/4], y \in [0, 1/4]$

In this case:

$$
\begin{aligned}
\mathsf{Average}_{n+1}(\eta(x), \eta(y)) &= \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y)))) \\
&= (\text{Diagram 3.2}) = \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))),
\end{aligned}
$$

by the inductive 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 $F_{n+1} = \mathsf{cons}_L(F_n)$ then $\mathsf{Average}_{n+1}(\eta(x), \eta(y))$ is of the form $\downarrow\mathsf{cons}_L(F_n)$. F_{n+1} is finite because F_n is.

Let $z \in F_{n+1}$. We must show that $\kappa_z \leq (\frac{4}{3})^{-n}$.

We know that $z = \mathsf{cons}_L(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq (\frac{4}{3})^{-n+1}$. We have: $z = \mathsf{cons}_L(t) = \frac{3t}{4}$; then:

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

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

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

Let $z \in F_{n+1}$. We must show that $\{z\} \sqsubseteq \eta(x \oplus y)$. As $z \in F_{n+1} = \widehat{\mathsf{cons}}_L(F_n)$, so $z = \mathsf{cons}_L(s)$ for some $s \in F_n$.

By the inductive hypothesis $\{s\} \sqsubseteq \eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_L(y))$, hence:

$$\begin{aligned}
\{z\} = \{\mathsf{cons}_L(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_L(\eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_L(y))) \\
(\text{definition of } \mathsf{tail}_L) &= \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{4x}{3} \oplus \frac{4y}{3}\right)\right) = \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{4x+4y}{6}\right)\right) \\
(\text{Diagram 3.1}) &= \eta\left(\mathsf{cons}_L\left(\frac{4x+4y}{6}\right)\right) = \eta\left(\left(\frac{3}{4}\right)\left(\frac{4x+4y}{6}\right)\right) \\
(\text{definition of } \mathsf{cons}_L) &= \eta\left(\frac{x+y}{2}\right) = \eta(x \oplus y).
\end{aligned}$$

as we wanted.

b) if $x \in [1/4, 3/4], y \in [1/4, 3/4]$, in this case:

$$\begin{aligned}
\mathsf{Average}_{n+1}(\eta(x), \eta(y)) = {} & \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y)))) \\
& \uplus \widehat{\mathsf{cons}}_C(\mathsf{Average}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_R(\eta(y)))) \\
& \uplus \widehat{\mathsf{cons}}_R(\mathsf{Average}_n(\widehat{\mathsf{tail}}_R(\eta(x)), \widehat{\mathsf{tail}}_R(\eta(y)))) \\
(\text{Diagram 3.2}) = {} & \widehat{\mathsf{cons}}_L(\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))) \\
& \uplus \widehat{\mathsf{cons}}_C(\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_R(y)))) \\
& \uplus \widehat{\mathsf{cons}}_R(\mathsf{Average}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_R(y)))),
\end{aligned}$$

by the inductive hypothesis,

- $\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))$ is of the form $\downarrow F_n'$ for F_n' finite.
- $\mathsf{Average}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_R(y)))$ is of the form $\downarrow F_n''$ for F_n'' finite.
- $\mathsf{Average}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_R(y)))$ is of the form $\downarrow F_n'''$ for F_n''' finite.

Take $F_{n+1} = \widehat{\mathsf{cons}}_L(F_n') \uplus \widehat{\mathsf{cons}}_C(F_n'') \uplus \widehat{\mathsf{cons}}_R(F_n''')$ then
$\mathsf{Average}_{n+1}(\eta(x), \eta(y))$ is of the form:

$$\downarrow\{\widehat{\mathsf{cons}}_L(F_n') \uplus \widehat{\mathsf{cons}}_C(F_n'') \uplus \widehat{\mathsf{cons}}_R(F_n''')\}.$$

F_{n+1} is finite because $F_n^{'}, F_n^{''}$ and $F_n^{'''}$ are.

Let $z \in F_{n+1}$ we must show that $\kappa_z \leq \frac{4}{3}^{-n}$. We have three cases:

i $z = \mathsf{cons}_L(t)$ for some $t \in F_n^{'}$,
ii $z = \mathsf{cons}_C(t)$ for some $t \in F_n^{''}$,
iii $z = \mathsf{cons}_R(t)$ for some $t \in F_n^{'''}$.

For the first case, we proceed as in case a).

For the second case, we have that $z = \frac{3t}{4} + \frac{1}{8} = \left[\frac{3\underline{t}}{4} + \frac{1}{8}, \frac{3\overline{t}}{4} + \frac{1}{8}\right]$ then $\kappa_z = \left(\frac{3}{4}\overline{t} + \frac{1}{8}\right) - \left(\frac{3}{4}\underline{t} + \frac{1}{8}\right) = \frac{3}{4}\overline{t} - \frac{3}{4}\underline{t}$. Again, we proceed as in case a). The third case is similar.

Let $z \in F_{n+1}$ we must show that $\{z\} \sqsubseteq \eta(x \oplus y)$. Again we have cases i), ii) and iii) as above.

For case i), we proceed as case a) above. For case ii), by the inductive hypothesis we know that $t \sqsubseteq \eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_R(y))$, hence:

$$\begin{aligned}
\{z\} = \{\mathsf{cons}_C(t)\} &\sqsubseteq \widehat{\mathsf{cons}}_C(\eta(\mathsf{tail}_L(x) \oplus \mathsf{tail}_R(y))) \\
\text{(definition of } \mathsf{tail}_L \text{ and } \mathsf{tail}_R) &= \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{4x}{3} \oplus \frac{4y-1}{3}\right)\right) \\
\text{(definition of } \oplus) &= \widehat{\mathsf{cons}}_C\left(\eta\left(\frac{4x+4y-1}{6}\right)\right) \\
\text{(Diagram 3.1)} &= \eta\left(\mathsf{cons}_C\left(\frac{x+y}{2} - \frac{1}{8}\right)\right) \\
\text{(definition of } \mathsf{cons}_C) &= \eta\left(\frac{x+y}{2}\right) = \eta(x \oplus y).
\end{aligned}$$

as we wanted. For case iii) the proof is similar to the previous one.

The proof is similar for the remaining cases. □

8.4 Multiplication

We now derive a recursive definition of a continuous extension of the multiplication map

$$mult : [0,1] \times [0,1] \to [0,1]$$

$$mult(x,y) \mapsto x \times y.$$

The extended map is given by $\mathsf{mult} : \mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I}$

$$
\begin{aligned}
\mathsf{mult}(X,Y) = \ & \text{if } \overline{\mathsf{rtest}}_{1/3,2/3}(X) \\
& \quad \text{then} \\
& \qquad \text{if } \overline{\mathsf{rtest}}_{1/3,2/3}(Y) \\
& \qquad\quad \text{then } \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_L(Y))) \\
& \qquad\quad \text{else } \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_R(Y))) + \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_L(X)\right) \\
& \quad \text{else} \\
& \qquad \text{if } \overline{\mathsf{rtest}}_{1/3,2/3}(Y) \\
& \qquad\quad \text{then } \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_L(Y))) + \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_L(Y)\right) \\
& \qquad\quad \text{else } \widehat{\mathsf{cons}}_{C_2}(\mathsf{mult}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_R(Y))) + \\
& \qquad\qquad\qquad \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_R(X) + \widehat{\mathsf{tail}}_R(Y)\right),
\end{aligned}
$$

Where $(\_ + \_) : \mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I}[0,2]$ is the continuous extension of the addition map (this extension can be easily derived from the average map). In the proof of Theorem 8.4.1, we work with the prefixed version of the map $(\_ + \_)$ which we call $\mathsf{plus}(\_,\_)$.

Let $L = [0, 2/3]$, $R = [1/3, 1]$, $C_1 = [0, 4/9]$ and $C_2 = [1/9, 5/9]$ hence the definitions of cons and tail are

$$
\begin{aligned}
\mathsf{cons}_L(x) &= \frac{2}{3}x, & \mathsf{tail}_L(x) &= \mathsf{min}\left\{\tfrac{3}{2}x, 1\right\}, \\
\mathsf{cons}_R(x) &= \frac{2x+1}{3}, & \mathsf{tail}_R(x) &= \mathsf{max}\left\{-0, \tfrac{3x-1}{2}\right\}, \\
\mathsf{cons}_{C_1}(x) &= \frac{4}{9}x, & \mathsf{tail}_{C_1}(x) &= \mathsf{min}\left\{\mathsf{max}\left\{0, \tfrac{9}{4}x\right\}, 1\right\}, \\
\mathsf{cons}_{C_2}(x) &= \frac{4x+1}{9}, & \mathsf{tail}_{C_2}(x) &= \mathsf{min}\left\{\mathsf{max}\left\{0, \tfrac{9x-1}{4}\right\}, 1\right\}.
\end{aligned}
$$

The analysis to obtain the above definition is the following:

(i) if $x \subseteq L$ and $y \subseteq L$ then $x \times y \subseteq C_1$,

(ii) if $x \subseteq L$ and $y \subseteq R$ then $x \times y \subseteq L$,

(iii) if $x \subseteq R$ and $y \subseteq L$ then $x \times y \subseteq L$,

(iv) if $x \subseteq R$ and $y \subseteq R$ then $x \times y \subseteq [1/9, 1]$.

For all $x, y \in [0,1]$ we have the equations:

$$\mathsf{cons}_L(x) \times \mathsf{cons}_L(y) = \frac{2}{3}x \times \frac{2}{3}y = \frac{4}{9}x \times y = \mathsf{cons}_{C_1}(x \times y).$$

$$\mathsf{cons}_L(x) \times \mathsf{cons}_R(y) = \frac{2}{3}x \times \frac{2y+1}{3} = \frac{4}{9}(x \times y) + \frac{2}{9}x = \mathsf{cons}_{C_1}(x \times y) + \frac{2}{9}x.$$

$$\mathsf{cons}_R(x) \times \mathsf{cons}_L(y) = \frac{2x+1}{3} \times \frac{2}{3}y = \frac{4}{9}(x \times y) + \frac{2}{9}y = \mathsf{cons}_{C_1}(x \times y) + \frac{2}{9}y.$$

$$\begin{aligned}\mathsf{cons}_R(x) \times \mathsf{cons}_R(y) \quad &= \tfrac{2x+1}{3} \times \tfrac{2y+1}{3} = \tfrac{4}{9}(x \times y) + \tfrac{2}{9}(x+y) + \tfrac{1}{9} = \\ &\qquad \mathsf{cons}_{C_2}(x \times y) + \tfrac{2}{9}(x+y).\end{aligned}$$

For maximal partial real numbers, this map coincides with the original map up to the identification of real numbers and maximal partial numbers.

Theorem 8.4.1. $\mathsf{mult}(\eta(x), \eta(y)) = \eta(x \times y)$ *for all* $x, y \in [0,1]$.

Proof. We take $D = (\mathcal{P}^H\mathcal{I} \times \mathcal{P}^H\mathcal{I} \to \mathcal{P}^H\mathcal{I})$. Define $\Phi : D \to D$ by, for all $\mathsf{F} \in D$,

$$\begin{aligned}\Phi(\mathsf{F})(X,Y) = \ &\mathsf{if}\ \overline{\mathsf{rtest}}_{0,2/3}(X) \\ &\quad \mathsf{then} \\ &\qquad \mathsf{if}\ \overline{\mathsf{rtest}}_{0,2/3}(Y) \\ &\qquad\quad \mathsf{then}\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{F}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_L(Y))) \\ &\qquad\quad \mathsf{else}\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{F}(\widehat{\mathsf{tail}}_L(X), \widehat{\mathsf{tail}}_R(Y))) + \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_L(X)\right) \\ &\quad \mathsf{else} \\ &\qquad \mathsf{if}\ \overline{\mathsf{rtest}}_{0,2/3}(Y) \\ &\qquad\quad \mathsf{then}\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{F}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_L(Y))) + \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_L(Y)\right) \\ &\qquad\quad \mathsf{else}\ \widehat{\mathsf{cons}}_{C_2}(\mathsf{F}(\widehat{\mathsf{tail}}_R(X), \widehat{\mathsf{tail}}_R(Y))) + \\ &\qquad\qquad \tfrac{2}{9}\left(\widehat{\mathsf{tail}}_R(X) + \widehat{\mathsf{tail}}_R(Y)\right),\end{aligned}$$

so that $\mathsf{mult} = \bigsqcup_n \mathsf{mult}_n$, where $\mathsf{mult}_n = \Phi^n(\bot)$. To conclude, we use Lemma 8.4.2 below. □

Lemma 8.4.2. *For any* $x, y \in [0,1]$*, the following conditions hold:*

1. $\mathsf{mult}_n(\eta(x), \eta(y))$ *is of the form* $\downarrow F_n$ *for* $F_n \subseteq \mathcal{I}[0,1]$ *finite,*
2. $\kappa_z \leq \left(\frac{3}{2}\right)^{-n+1}$ *for each* $z \in F_n$*,*
3. $F_n \sqsubseteq \eta(x \times y)$.

Proof. The proof is by induction on the number of unfoldings n.

If $n = 0$, we know that $\mathsf{mult}_0(\eta(x), \eta(y)) = \{\bot\} = \downarrow\{\bot\}$ for any $x, y \in [0,1]$. Take $z \in F_n = \{\bot\}$, so $\kappa_z = 1 < (3/2)$, and $\{\bot\} \sqsubseteq \eta(x \times y)$ for all $x, y \in [0,1]$.

Assume that it is true for n. We prove that it is true for $n+1$. We proceed according to the positions of x and y.

1. if $x \in L, y \in L$ we have three cases:

   a) if $x \in [0, 1/3], y \in [0, 1/3]$, in this case:

$$\begin{aligned}\mathsf{mult}_{n+1}(\eta(x), \eta(y)) &= \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y)))) \\ \text{(Diagram 3.2)} &= \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))),\end{aligned}$$

   by the inductive hypothesis, $\mathsf{mult}_n(\eta(t), \eta(s))$ is of the form $\downarrow F_n$ for F_n finite, $t = \mathsf{tail}_L(x)$ and $s = \mathsf{tail}_L(y)$. Take $F_{n+1} = \mathsf{cons}_{C_1}(F_n)$ then $\mathsf{mult}_{n+1}(\eta(x), \eta(y))$ is of the form $\downarrow \mathsf{cons}_{C_1}(F_n)$. F_{n+1} is finite because F_n is.

   Let $z \in F_{n+1}$. We must show that $\kappa_z \leq (\frac{3}{2})^{-n}$. We know that $z = \mathsf{cons}_{C_1}(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq (\frac{3}{2})^{-n+1}$. We have: $z = \mathsf{cons}_{C_1}(t) = \frac{4t}{9}$; then:

$$\overline{t} - \underline{t} \leq \left(\frac{3}{2}\right)^{-n+1},$$

$$\frac{4}{9}\overline{t} - \frac{4}{9}\underline{t} \leq \left(\frac{4}{9}\right)\left(\frac{3}{2}\right)^{-n+1} = \left(\frac{3}{2}\right)^{-n-1} < \left(\frac{3}{2}\right)^{-n},$$

   hence $\kappa_z \leq (\frac{3}{2})^{-n}$.

   Let $z \in F_{n+1}$. We must show that $\{z\} \sqsubseteq \eta(x \times y)$. As $z \in F_{n+1} = \widehat{\mathsf{cons}}_{C_1}(F_n)$, hence $z = \mathsf{cons}_{C_1}(s)$ for some $s \in F_n$. By the inductive hypothesis $\{s\} \sqsubseteq \eta(\mathsf{tail}_L(x) \times \mathsf{tail}_L(y))$, hence:

$$\begin{aligned}\{z\} = \{\mathsf{cons}_{C_1}(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_{C_1}(\eta(\mathsf{tail}_L(x) \times \mathsf{tail}_L(y))) \\ \text{(definition of } \mathsf{tail}_L) &= \widehat{\mathsf{cons}}_{C_1}\left(\eta\left(\frac{3x}{2} \times \frac{3y}{2}\right)\right) = \widehat{\mathsf{cons}}_{C_1}\left(\eta\left(\frac{9xy}{4}\right)\right) \\ \text{(Diagram 3.1)} &= \eta\left(\mathsf{cons}_{C_1}\left(\frac{9xy}{4}\right)\right) = \eta\left(x \times y\right).\end{aligned}$$

   as we wanted.

b) if $x \in [1/3, 2/3], y \in [1/3, 2/3]$, in this case:

$$\begin{aligned}
&\mathsf{mult}_{n+1}(\eta(x), \eta(y)) = \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y))))\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\widehat{\mathsf{tail}}_L(\eta(x)), \widehat{\mathsf{tail}}_R(\eta(y)))) + \frac{2}{9}\left(\widehat{\mathsf{tail}}_L\left(\eta\left(x\right)\right)\right)\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\widehat{\mathsf{tail}}_R(\eta(x)), \widehat{\mathsf{tail}}_L(\eta(y)))) + \frac{2}{9}\left(\widehat{\mathsf{tail}}_L\left(\eta\left(y\right)\right)\right)\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{C_2}(\mathsf{mult}_n(\widehat{\mathsf{tail}}_R(\eta(x)), \widehat{\mathsf{tail}}_R(\eta(y)))) +\\
&\qquad\qquad \frac{2}{9}\left(\widehat{\mathsf{tail}}_R\left(\eta\left(x\right)\right) + \widehat{\mathsf{tail}}_R\left(\eta\left(y\right)\right)\right)\\
&= \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y))))\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_R(y)))) + \frac{2}{9}\left(\eta\left(\mathsf{tail}_L(x)\right)\right)\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{C_1}(\mathsf{mult}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_L(y)))) + \frac{2}{9}\left(\eta\left(\mathsf{tail}_L(x)\right)\right)\\
&\quad \uplus\ \widehat{\mathsf{cons}}_{R}(\mathsf{mult}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_R(y)))) +\\
&\qquad\qquad \frac{2}{9}\left(\eta\left(\mathsf{tail}_L(x)\right) + \eta\left(\mathsf{tail}_L(x)\right)\right),
\end{aligned}$$

by the inductive hypothesis,

- $\mathsf{mult}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_L(y)))$ is of the form $\downarrow F_n^{'}$ for $F_n^{'}$ finite.
- $\mathsf{mult}_n(\eta(\mathsf{tail}_L(x)), \eta(\mathsf{tail}_R(y)))$ is of the form $\downarrow F_n^{''}$ for $F_n^{''}$ finite.
- $\mathsf{mult}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_L(y)))$ is of the form $\downarrow F_n^{'''}$ for $F_n^{'''}$ finite.
- $\mathsf{mult}_n(\eta(\mathsf{tail}_R(x)), \eta(\mathsf{tail}_R(y)))$ is of the form $\downarrow F_n^{iv}$ for F_n^{iv} finite.

Take $F_{n+1} = \widehat{\mathsf{cons}}_{C_1}(F_n^{'}) \uplus \widehat{\mathsf{cons}}_{C_1}(F_n^{''}) + \frac{2}{9}\widehat{\mathsf{tail}}_L(x) \uplus \widehat{\mathsf{cons}}_{C_1}(F_n^{'''}) + \frac{2}{9}\widehat{\mathsf{tail}}_L(y) \uplus \widehat{\mathsf{cons}}_{C_2}(F_n^{iv})$ then $\mathsf{mult}_{n+1}(\eta(x), \eta(y))$ is of the form:

$$\begin{aligned}
&\downarrow\{\widehat{\mathsf{cons}}_{C_1}(F_n^{'}) \uplus \widehat{\mathsf{cons}}_{C_1}(F_n^{''}) + \frac{2}{9}\widehat{\mathsf{tail}}_L(x) \uplus\\
&\quad \widehat{\mathsf{cons}}_{C_1}(F_n^{'''}) + \tfrac{2}{9}\widehat{\mathsf{tail}}_L(y) \uplus \widehat{\mathsf{cons}}_{C_2}(F_n^{iv}) + \tfrac{2}{9}(\widehat{\mathsf{tail}}_L(x) + \widehat{\mathsf{tail}}_L(y))\}.
\end{aligned}$$

F_{n+1} is finite because $F_n^{'}, F_n^{''}, F_n^{'''}$ and F_n^{iv} are.

Let $z \in F_{n+1}$ we must show that $\kappa_z \leq \frac{3}{2}^{-n}$. We have four cases:

i $z = \mathsf{cons}_{C_1}(t)$ for some $t \in F_n^{'}$,

ii $z = \mathsf{cons}_{C_1}(t) + \frac{2}{9}\mathsf{tail}_L(x)$ for some $t \in F_n^{''}$,

iii $z = \mathsf{cons}_{C_1}(t) + \frac{2}{9}\mathsf{tail}_L(y)$ for some $t \in F_n^{'''}$,

iv $z = \mathsf{cons}_{C_2}(t) + \frac{2}{9}(\mathsf{tail}_L(x) + \mathsf{tail}_L(y))$ for some $t \in F_n^{iv}$.

For the first case, we proceed as in case a). For the second case, we know that x is maximal, so the length of x is 0, so we have case one. For the third case, the same argument applies since y is maximal. The fourth case

$$\overline{t} - \underline{t} \leq \left(\frac{3}{2}\right)^{-n+1}$$

$$\frac{4\overline{t}+1}{9} - \frac{4\underline{t}+1}{9} \leq \left(\frac{4}{9}\right)\left(\frac{3}{2}\right)^{-n+1} = \left(\frac{3}{2}\right)^{-n-1} < \left(\frac{3}{2}\right)^{-n},$$

hence $\kappa_z \leq (\frac{3}{2})^{-n}$.

Let $z \in F_{n+1}$ we must show that $\{z\} \sqsubseteq \eta(x \times y)$. Again we have cases i,ii,iii and iv as above.

For case i), we proceed as case a) above.

For case ii), by the inductive hypothesis we know that $t \sqsubseteq \eta(\mathsf{tail}_L(x) \times \mathsf{tail}_R(y))$, hence:

$$\begin{aligned}
\{z\} &= \{\mathsf{cons}_{C_2}(t) + \frac{1}{3}(x)\} \\
&\sqsubseteq \mathsf{plus}\left(\widehat{\mathsf{cons}}_{C_2}(\eta(\mathsf{tail}_L(x) \times \mathsf{tail}_R(y))), \eta\left(\frac{x}{3}\right)\right) \\
\text{(definition of } \mathsf{tail}) &= \mathsf{plus}\left(\widehat{\mathsf{cons}}_{C_2}\left(\eta\left(\frac{3x}{2} \times \frac{3y-1}{2}\right)\right), \eta\left(\frac{x}{3}\right)\right) \\
\text{(definition of } \times) &= \mathsf{plus}\left(\widehat{\mathsf{cons}}_{C_2}\left(\eta\left(\frac{9x \times y - 3x}{4}\right)\right), \eta\left(\frac{x}{3}\right)\right) \\
\text{(Diagram 3.1)} &= \mathsf{plus}\left(\eta\left(\mathsf{cons}_{C_2}\left(\frac{9(x \times y)}{4}\right)\right), \eta\left(\frac{x}{3}\right)\right) \\
\text{(definition of } \mathsf{cons}_{C_1}) &= \mathsf{plus}\left(\eta\left((x \times y) - \frac{x}{3}\right), \eta\left(\frac{x}{3}\right)\right) \\
\text{(definition of } \mathsf{plus}) &= \eta\left((x \times y) - \frac{x}{3} + \frac{x}{3}\right) \\
&= \eta(x \times y).
\end{aligned}$$

as we wanted. Cases iii and iv follow similarly.

The rest of the proof is similar to cases a) and b). □

8.5 Division of two real numbers

We give a definition of division of two real numbers. We use Plume's [69] algorithm for defining division. To simplify discussion, we define division in the following intervals:

$$div : [-1, 1] \times [1/4, 1] \to [-4, 4]$$

In that sense we give a definition for $div(x, 4y)$ to keep the result in $[-1, 1]$. For example $div\left(\frac{1}{4}, \frac{3}{8}\right)$ should produce as a result $\frac{\frac{1}{4}}{4\frac{3}{8}} = \frac{\frac{1}{4}}{\frac{3}{2}} = \frac{1}{6}$. Take

$$L = [-1, 0], \quad C = [-1/2, 1/2], \quad R = [0, 1],$$

$$C_1 = [-1/4, 3/4], \quad C_2 = [-3/8, 5/8], \quad C_3 = [-3/4, 1/4], \quad C_4 = [-5/8, 3/8],$$

hence:

$$\mathsf{cons}_L(x) = \frac{x-1}{2}, \qquad \mathsf{tail}_L(x) = \mathsf{min}\{2x+1, 1\},$$

$$\mathsf{cons}_C(x) = \frac{x}{2}, \qquad \mathsf{tail}_C(x) = \mathsf{max}\{-1, \mathsf{min}\{2x, 1\}\},$$

$$\mathsf{cons}_R(x) = \frac{x+1}{2}, \qquad \mathsf{tail}_R(x) = \mathsf{max}\{-1, 2x-1\},$$

$$\mathsf{cons}_{C_1}(x) = \frac{x}{2} + \frac{1}{4}, \qquad \mathsf{tail}_{C_1}(x) = \mathsf{min}\{\mathsf{max}\{-1, 2x - \frac{1}{2}\}, 1\},$$

$$\mathsf{cons}_{C_2}(x) = \frac{x}{2} + \frac{1}{8}, \qquad \mathsf{tail}_{C_2}(x) = \mathsf{min}\{\mathsf{max}\{-1, 2x - \frac{1}{4}\}, 1\},$$

$$\mathsf{cons}_{C_3}(x) = \frac{x}{2} - \frac{1}{4}, \qquad \mathsf{tail}_{C_3}(x) = \mathsf{max}\{-1, \mathsf{min}\{2x + \frac{1}{2}, 1\}\},$$

$$\mathsf{cons}_{C_4}(x) = \frac{x}{2} - \frac{1}{8}, \qquad \mathsf{tail}_{C_4}(x) = \mathsf{max}\{-1, \mathsf{min}\{2x + \frac{1}{4}, 1\}\},$$

The definition of division is the following:

$$\mathsf{div} : \mathcal{P}^H\mathcal{I}[-1, 1] \times \mathcal{P}^H\mathcal{I}[1/4, 1] \to \mathcal{P}^H\mathcal{I}[-1, 1]$$

$$
\begin{array}{l}
\mathsf{div}(X,Y) = \\
\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,1/2}(X) \\
\qquad \mathsf{then} \\
\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,-1/4}(X) \\
\qquad\qquad \mathsf{then} \\
\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,1/2}(X+Y) \\
\qquad\qquad\qquad \mathsf{then} \\
\qquad\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,-1/4}(X+Y) \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \ \widehat{\mathsf{cons}}_L(\mathsf{div}(\widehat{\mathsf{tail}}_C(X+2Y),Y)) \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \ \widehat{\mathsf{cons}}_{C_3}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X+Y),Y)) \\
\qquad\qquad\qquad \mathsf{else} \\
\qquad\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{1/4,1/2}(X+Y) \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \ \widehat{\mathsf{cons}}_{C_3}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X+Y),Y)) \\
\qquad\qquad\qquad\qquad\ \ \mathsf{else}\ \ \widehat{\mathsf{cons}}_{C_4}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X)+Y,Y)) \\
\qquad\qquad \mathsf{else}\ \ \widehat{\mathsf{cons}}_C(\mathsf{div}(\widehat{\mathsf{tail}}_C(X),Y)) \\
\qquad \mathsf{else} \\
\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{1/4,1/2}(X) \\
\qquad\qquad \mathsf{then}\ \ \widehat{\mathsf{cons}}_C(\mathsf{div}(\widehat{\mathsf{tail}}_C(X),Y)) \\
\qquad\qquad \mathsf{else} \\
\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,1/2}(X-Y) \\
\qquad\qquad\qquad \mathsf{then} \\
\qquad\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{-1/2,-1/4}(X-Y) \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \ \widehat{\mathsf{cons}}_{C_2}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X)-Y,Y)) \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \ \widehat{\mathsf{cons}}_{C_1}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X-Y),Y)) \\
\qquad\qquad\qquad \mathsf{else} \\
\qquad\qquad\qquad\quad \mathsf{if}\ \ \overline{\mathsf{rtest}}_{1/4,1/2}(X-Y) \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \ \widehat{\mathsf{cons}}_{C_1}(\mathsf{div}(\widehat{\mathsf{tail}}_C(X-Y),Y)) \\
\qquad\qquad\qquad\qquad\ \ \mathsf{else}\ \ \widehat{\mathsf{cons}}_R(\mathsf{div}(\widehat{\mathsf{tail}}_C(X-2Y),Y))
\end{array}
$$

For maximal partial elements we show that it indeed coincides with the original map.

Theorem 8.5.1. $\mathsf{div}(\eta(x),\eta(y)) = \eta\left(\frac{x}{4y}\right)$ *for all* $x \in [-1,1]$ *and* $y \in [1/4,1]$.

Proof. We take $D = (\mathcal{P}^H\mathcal{I}[-1,1] \times \mathcal{P}^H\mathcal{I}[1/4,1] \to \mathcal{P}^H\mathcal{I}[-1,1])$. Define $\Phi : D \to D$ by, for all $\mathsf{F} \in D$,

$\Phi(\mathsf{F})(X,Y) = \ \mathsf{div}(X,Y)$

so that $\mathsf{div} = \bigsqcup_n \mathsf{div}_n$, where $\mathsf{div}_n = \Phi^n(\bot)$. To conclude, we use Lemma 8.5.2. □

Lemma 8.5.2. *For any $x \in [-1,1]$ and $y \in [1/4,1]$, the following conditions hold:*

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

Proof. The proof is by induction on the number of unfolding n.

If $n = 0$, we know that $\mathsf{div}_0(\eta(x),\eta(y)) = \{\bot\} = \downarrow\{\bot\}$ for any $x \in [-1,1]$ and $y \in [1/4,1]$. Take $z \in F_n = \{\bot\}$, so $\kappa_z = 2$, and $\{\bot\} \sqsubseteq \eta(x/4y)$ for all $x \in [-1,1]$ and $y \in [1/4,1]$.

Assume that it is true for n. We prove that it is true for $n+1$. We proceed according to the positions of x and y. We present only two cases. The reminder cases can be shown by a careful, but routine, modification of the previous cases.

1. if $x \in [-1,-1/2]$ and $x + y \in [-1,-1/2]$ we have that:

$$\begin{aligned}\mathsf{div}_{n+1}(\eta(x),\eta(y)) &= \widehat{\mathsf{cons}}_L(\mathsf{div}_n(\widehat{\mathsf{tail}}_C(\eta(x)+2\eta(y)),\eta(y)))\\ \text{(Diagram 3.2)} &= \widehat{\mathsf{cons}}_L(\mathsf{div}_n(\eta(\mathsf{tail}_C(x+2y)),\eta(y))),\end{aligned}$$

by the inductive hypothesis, $\mathsf{div}_n(\eta(t),\eta(s))$ is of the form $\downarrow F_n$ for F_n finite, $t = \mathsf{tail}_C(x+2y)$ and $s = y$. Take $F_{n+1} = \mathsf{cons}_L(F_n)$ then $\mathsf{div}_{n+1}(\eta(x),\eta(y))$ is of the form $\downarrow\mathsf{cons}_L(F_n)$. F_{n+1} is finite because F_n is.

Let $z \in F_{n+1}$. We must show that $\kappa_z \leq (2)^{-n}$.

We know that $z = \mathsf{cons}_L(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq (2)^{-n+1}$. We have: $z = \mathsf{cons}_L(t) = \frac{t-1}{2}$; then:

$$\overline{t} - \underline{t} \leq (2)^{-n+1}$$

$$\frac{\overline{t}}{2} - \frac{\underline{t}}{2} \leq \left(\frac{1}{2}\right)(2)^{-n+1} = (2)^{-n}$$

hence $\kappa_z \leq (2)^{-n}$.

Let $z \in F_{n+1}$. We must show that $\{z\} \sqsubseteq \eta(x/4y)$. $z \in F_{n+1} = \widehat{\mathsf{cons}}_L(F_n)$, so $z = \mathsf{cons}_L(s)$ for some $s \in F_n$.

By the inductive hypothesis $\{s\} \sqsubseteq \eta(\mathsf{tail}_C(x+2y)/4y)$, hence:

$$\begin{aligned}\{z\} = \{\mathsf{cons}_L(s)\} &\sqsubseteq \widehat{\mathsf{cons}}_L(\eta(\mathsf{tail}_C(x+2y)/4y)) \\ (\text{definition of } \mathsf{tail}_C) &= \widehat{\mathsf{cons}}_L\left(\eta\left(\frac{2x+4y}{4y}\right)\right) \\ (\text{Diagram 3.1}) &= \eta\left(\mathsf{cons}_L\left(\frac{x}{2y}+1\right)\right) \\ (\text{definition of } \mathsf{cons}_L) &= \eta\left(\frac{x}{4y}\right).\end{aligned}$$

as we wanted.

2. if $x \in [1/2, 1], x - y \in [1/2, 1]$, in this case:

$$\begin{aligned}\mathsf{div}_{n+1}(\eta(x), \eta(y)) &= \widehat{\mathsf{cons}}_R(\mathsf{div}_n(\widehat{\mathsf{tail}}_C(\eta(x) - 2\eta(y)), \eta(y))) \\ (\text{Diagram 3.2}) &= \widehat{\mathsf{cons}}_R(\mathsf{div}_n(\eta(\mathsf{tail}_C(x-2y)), \eta(y))),\end{aligned}$$

by the inductive hypothesis, $\mathsf{div}_n(\eta(t), \eta(s))$ is of the form $\downarrow F_n$ for F_n finite, $t = \mathsf{tail}_C(x-2y)$ and $s = y$. Take $F_{n+1} = \widehat{\mathsf{cons}}_R(F_n)$ then $\mathsf{div}_{n+1}(\eta(x), \eta(y))$ is of the form $\downarrow\widehat{\mathsf{cons}}_R(F_n)$. F_{n+1} is finite because F_n is.

Let $z \in F_{n+1}$. We must show that $\kappa_z \leq (2)^{-n}$.

We know that $z = \mathsf{cons}_R(t)$ for some $t \in F_n$. By the inductive hypothesis $\kappa_t \leq (2)^{-n+1}$. We have: $z = \mathsf{cons}_R(t) = \frac{t+1}{2}$; then:

$$\overline{t} - \underline{t} \leq (2)^{-n+1}$$

$$\frac{\overline{t}+1}{2} - \frac{\underline{t}+1}{2} \leq \left(\frac{1}{2}\right)(2)^{-n+1} = (2)^{-n}$$

hence $\kappa_z \leq (2)^{-n}$.

Let $z \in F_{n+1}$. We must show that $\{z\} \sqsubseteq \eta(x/4y)$. $z \in F_{n+1} = \widehat{\mathsf{cons}}_R(F_n)$, so $z = \mathsf{cons}_R(s)$ for some $s \in F_n$.

By the inductive hypothesis $\{s\} \sqsubseteq \eta(\mathsf{tail}_C(x-2y)/4y)$, hence:

$$\begin{aligned}\{z\} = \{\mathsf{cons}_L(R)\} &\sqsubseteq \widehat{\mathsf{cons}}_R(\eta(\mathsf{tail}_C(x-2y)/4y)) \\ (\text{definition of } \mathsf{tail}_C) &= \widehat{\mathsf{cons}}_R\left(\eta\left(\frac{2x-4y}{4y}\right)\right) \\ (\text{Diagram 3.1}) &= \eta\left(\mathsf{cons}_R\left(\frac{x}{2y}-1\right)\right) \\ (\text{definition of } \mathsf{cons}_R) &= \eta\left(\frac{x}{4y}\right).\end{aligned}$$

as we wanted. The rest of the proof follows similarly.

□

8.6 Upper integer bound

We present a recursive definition of the continuous extension of the computable upper integer bound function presented in Section 6.2.

The extended map is given by

$$\mathsf{upper\_bound} : \mathcal{P}^H\mathcal{R} \to \mathcal{P}^H\mathbb{N}_\perp$$

$$\begin{array}{l}
\mathsf{upper_bound}(X) = \ \mathsf{if}\ \overline{\mathsf{rtest}}_{a,b}(X) \\
\quad \mathsf{then} \\
\quad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{a,0}(X) \\
\quad\quad\quad \mathsf{then} \\
\quad\quad\quad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{-1,a}(X) \\
\quad\quad\quad\quad\quad \mathsf{then}\ (\mathsf{upper_bound}(x+1)) - 1 \\
\quad\quad\quad\quad\quad \mathsf{else}\ 0 \\
\quad\quad\quad \mathsf{else}\ 1 \\
\quad \mathsf{else} \\
\quad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{0,b}(X) \\
\quad\quad\quad \mathsf{then}\ 1 \\
\quad\quad\quad \mathsf{else} \\
\quad\quad\quad\quad \mathsf{if}\ \overline{\mathsf{rtest}}_{b,1}(X) \\
\quad\quad\quad\quad\quad \mathsf{then}\ 1 \\
\quad\quad\quad\quad\quad \mathsf{else}\ (\mathsf{upper_bound}(x-1)) + 1
\end{array}$$

where $a, b \in \mathbb{Q}$ such that $-1 < a < 0 < b < 1$.

It is easy to show that for maximal partial numbers, this definition indeed coincides with the original map.

Part III

A Programming Language for Sequential Computation over the Reals

Chapter 9

The Programming Language *LRT*

In this Chapter, we introduce the **L**anguage for **R**edundant **T**est (LRT), which amounts to the language considered by Escardó [22] with the parallel conditional and the head construction removed and the sequential conditional and a constant $\mathsf{rtest}_{a,b}$ 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 further work.

In Section 9.1 we present the syntax of the language. The denotational semantics based on the Hoare powerdomain is presented in Section 9.2. In Section 9.3 the operational semantics of the language is presented. We show in Section 9.4 that the operational semantics is computationally adequate with respect to the denotational semantics given.

9.1 Syntax

The *language for redundant test LRT* is an extension of $\mathcal{L}_{PCF}$ with a ground type for real numbers and suitable primitive functions for real-number computation. Its raw syntax is given by:

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

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, b and c to denote numbers and intervals in different contexts.) Terms of ground type `I` are intended to compute real numbers in the unit interval. Programs constructed with this type are known as real programs.

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

9.2 Denotational Semantics

We denote by $\mathcal{A}$ the standard interpretation of LRT. 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:

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

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

The *interpretation* of the language LRT is a collection of domains $\{D_\sigma\}_\sigma$ together with a mapping:

$$c \mapsto \mathcal{A}[\![c]\!] : LRT \to \bigcup_\sigma \{D_\sigma\}$$

which is type-respecting in the sense that if $c : \sigma$ then $\mathcal{A}[\![c]\!] \in D_\sigma$. We write $[\![c]\!]$ instead of $\mathcal{A}[\![c]\!]$ for simplicity.

The interpretation of constants in LRT is defined as follows:

1. $[\![\texttt{true}]\!] = \eta(\mathsf{true})$;
2. $[\![\texttt{false}]\!] = \eta(\mathsf{false})$;
3. $[\![\texttt{k}_\texttt{n}]\!] = \eta(n)$;
4. $[\![(+1)]\!] = \widehat{(+1)}$,
$$\widehat{(+1)}X = \mathsf{cl}\{(+1)x \mid x \in X\};$$
5. $[\![(-1)]\!] = \widehat{(-1)}$,
$$\widehat{(-1)}X = \mathsf{cl}\{(-1)x \mid x \in X\};$$

6. $[\![(=0)]\!] = \widehat{(=0)}$,

$$\widehat{(=0)}X = \mathsf{cl}\{(=0)x \mid x \in X\};$$

7. $[\![\mathtt{cons}_a]\!] = \widehat{\mathsf{cons}_a}$,

$$\widehat{\mathsf{cons}_a}\ N = \mathsf{cl}\{\mathsf{cons}_a n \mid n \in N\};$$

8. $[\![\mathtt{tail}_a]\!] = \widehat{\mathsf{tail}_a}$,

$$\widehat{\mathsf{tail}_a}\ N = \mathsf{cl}\{\mathsf{tail}_a n \mid n \in N\};$$

9. $[\![\mathtt{Y}_\sigma]\!](F) = \bigsqcup_{n\geq 0} F^n(\{\bot\})$ $\left(F^n \in D_{(\sigma\to\sigma)}\right)$,

10.

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

11. $[\![\mathtt{rtest}_{a,b}]\!] = \overline{\mathsf{rtest}}_{a,b}$,

$$\overline{\mathsf{rtest}}_{a,b}M = \mathsf{cl} \bigcup_{m\in M} \mathrm{rtest}_{a,b}\, m,$$

where $\mathrm{rtest}_{a,b}$ is defined as:

$$\mathrm{rtest}_{a,b}(x) = \begin{cases} \downarrow\{\mathsf{tt},\mathsf{ff}\}, & \text{if } a < \underline{x} < \overline{x} < b; \\ \downarrow\{\mathsf{tt}\}, & \text{if } \overline{x} \leq b; \\ \downarrow\{\mathsf{ff}\}, & \text{if } \underline{x} \geq a; \\ \{\bot\}, & \text{otherwise.} \end{cases}$$

The functions $\eta, \widehat{\ }, \overline{\ }$ are defined in Section 2.5, the functions $(+1)$, (-1), $(=0)$ are the standard interpretations in the Scott model of PCF (Section 5.1), the functions $\mathsf{cons}_a, \mathsf{tail}_a$ are defined in Section 5.2, and the function $\mathsf{rtest}_{a,b}$ is defined in Chapter 7. The function $[\![\mathsf{Y}_\sigma]\!]$ is the least fixed point operator defined in Section 2.3.1.

9.3 Operational Semantics

We consider a small-step style operational semantics for our language. We define the one-step reduction relation $\rightarrow$ to be the least relation satisfying the one-step reduction rules for evaluation of PCF and those below:

1. $\mathtt{cons}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{ab} M$	
2. $\mathtt{cons}_a M \rightarrow \mathtt{cons}_a M'$	if $M \rightarrow M'$
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 $b > a$
5. $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{b \setminus a} M$	if $a \sqsubseteq b$ and $a \neq b$
6. $\mathtt{tail}_a(\mathtt{cons}_b M) \rightarrow \mathtt{cons}_{(a \sqcup b) \setminus a}(\mathtt{tail}_{(a \sqcup b) \setminus b} M)$	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 $M \rightarrow M'$
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 $M \rightarrow M'$
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 9.3.1.

- Rules 2, 7, and 13 can be applied infinitely often leading to a divergent computation. This is consistent with the Hoare semantics which always allows $\perp$ as a possible denotational value of a term.

- Rules 11-13 cannot be made deterministic given the particular computational adequacy formulation which is proved in Section 9.4. This means that the set of rewrite rules needs to be 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* (see Chapter 10 for a further discussion). On the other hand, in the absense of $\mathtt{rtest}_{a,b}$, the rules 1-10 can be made deterministic without loss of computational adequacy if one so desires.

- In practice, one would like to avoid divergent computations by considering a strategy of the application for the rules. This is the topic of Chapter 10 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 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 9.3.2. *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 $M : \mathtt{I}$, then we define

$$[M]_i = \quad \biguplus \{\eta(a) \mid \exists M' \exists k \leq i, M \stackrel{k}{\to} \mathtt{cons}_a M'\}$$

if this set is non-empty. Here the relation $\stackrel{k}{\to}$ denotes the k-fold composition of the relation $\to$.

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

$$[M]_i = \quad \biguplus \{\eta(n) \mid \exists k \leq i, M \stackrel{k}{\to} \mathtt{k_n}\}$$

if this set is non-empty.

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

$$[M]_i = \quad \biguplus \{\eta(b) \mid \exists k \leq i, M \stackrel{k}{\to} b, b \in \{\mathtt{true}, \mathtt{false}\}\}$$

if this set is non-empty.

The following is immediate

Proposition 9.3.3. $[M]_i \sqsubseteq [M]_{i+1}$.

Definition 9.3.4.

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

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.

9.4 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 defined in Section 9.3 and $[\![M]\!]$ is the denotational meaning of M defined in Section 9.2.

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

Lemma 9.4.1. $[\![M]\!] = \biguplus \{[\![N]\!] \mid M \to N\}$ *(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 $M \equiv (-1)M'$ and $M \to N$, there are three rules that apply to predecessor.

First case: $M \equiv (-1)\mathtt{k}_0$ and $(-1)\mathtt{k}_0 \to \mathtt{k}_0 \equiv N$,

$$\begin{array}{rcl} [\![(-1)\mathtt{k}_0]\!] & = & \widehat{(-1)}[\![\mathtt{k}_0]\!] = \widehat{(-1)}\{0, \bot\} = \mathsf{cl}\{(-1)0, (-1)\bot\} \\ & = & \mathsf{cl}\{0, \bot\} = \{0, \bot\} = [\![\mathtt{k}_0]\!] = [\![N]\!]. \end{array}$$

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

$$\begin{array}{rcl} [\![(-1)\mathtt{k}_{n+1}]\!] & = & \widehat{(-1)}([\![\mathtt{k}_{n+1}]\!]) = \widehat{(-1)}\{n+1, \bot\} = \mathsf{cl}\{(-1)n+1, (-1)\bot\} \\ & = & \mathsf{cl}\{n, \bot\} = \{n, \bot\} = [\![\mathtt{k}_n]\!] = [\![N]\!]. \end{array}$$

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

$$\begin{array}{rcl} [\![M]\!] = [\![(-1)M']\!] & = & \widehat{(-1)}[\![M']\!] = \widehat{(-1)}\,(\biguplus \{[\![N']\!] \mid M' \to N'\}) \\ & = & \biguplus \{\widehat{(-1)}[\![N']\!] \mid M' \to N'\} \\ & = & \biguplus \{[\![(-1)N']\!] \mid M' \to N'\}, \end{array}$$

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

$$\begin{array}{rcl} [\![M]\!] & = & \overline{\mathsf{rtest}}_{p,q}\,(\biguplus \{[\![N']\!] \mid M' \to N'\}) \\ & = & \biguplus \{\overline{\mathsf{rtest}}_{p,q}[\![N']\!] \mid M' \to N'\} \\ & = & \biguplus \{[\![\mathtt{rtest}_{p,q}N']\!] \mid M' \to N'\} \end{array}$$

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 $\mathtt{rtest}_{p,q}(\mathtt{cons}_a(M''))$ where M'' is not a $\mathtt{cons}_a$ term. 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:

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

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

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

So the last expression exhausts the terms that are single-step derivable from M.

Third case: M is of the form $\mathtt{rtest}_{p,q}(\mathtt{cons}_a(\mathtt{cons}_b M'''))$ where M''' is not a $\mathtt{cons}$ term. Note that $[\![M]\!] = [\![\mathtt{rtest}_{p,q}(\mathtt{cons}_{ab}(M'''))]\!]$. Because $\uplus$ is idempotent, we get throw this term in as well:

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

And again, these exhaust the terms that are single-step derivable from M.

□

Lemma 9.4.2. $\forall j, [\![M]\!] = \uplus\,\{[\![N]\!] \mid M \xrightarrow{j} N\}$.

Proof. We proceed by induction on the length of the evaluation j. If the length is zero, then it follows trivially.

Suppose that it follows for j-folds and we want to prove that it holds for $j+1$-folds. Assume that $M \xrightarrow{j+1} N_1, M \xrightarrow{j+1} N_2, \cdots, M \xrightarrow{j+1} N_n$, then there are $N_1', N_2', \ldots, N_n'$ such that $M \xrightarrow{j} N_1' \to N_1, M \xrightarrow{j} N_2' \to N_2, \ldots, M \xrightarrow{j} N_n' \to N_n$. By assumption $[\![M]\!] = \uplus\,[\![N_i']\!], 0 \le i \le n$. By Lemma 9.4.1 by induction on the structure of M, we must therefore have $[\![M]\!] = \uplus\,[\![N_i]\!], 0 \le i \le n$. □

To prove soundness of the operational semantics we state the following Lemma:

Lemma 9.4.3. *For all programs M, $[M]_i \sqsubseteq [\![M]\!] = \biguplus \{[\![N]\!] \mid M \xrightarrow{i} N\}$.*

Proof. Let $b \in [M]_i, b \neq \bot$. By definition, $b \sqsubseteq a$ for some a and M' such that $M \xrightarrow{i} \mathsf{cons}_a M'$. $b \sqsubseteq a \sqsubseteq \mathsf{cons}_a x, \forall x \in [\![M']\!]$. As $\widehat{\mathsf{cons}_a}[\![M']\!] \sqsubseteq [\![\mathsf{cons}_a M']\!]$, by Lemma 9.4.2, $b \in \downarrow[\![\mathsf{cons}_a M']\!]$ then $b \in [\![M]\!]$. □

Proposition 9.4.4. ***(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 $M \xrightarrow{i} \mathsf{cons}_a M'$. Because $\widehat{\mathsf{cons}_a}[\![M']\!] = [\![\mathsf{cons}_a M']\!]$, Lemma 9.4.3 shows that $b \in \downarrow[\![\mathsf{cons}_a M']\!]$. 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 work with a notion of computability as in [68, 22].

Definition 9.4.5. *The set of computable terms is the least collection of terms such that:*

- *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 $x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n$ 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 9.4.6. *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. □

Theorem 9.4.7. *(**Completeness**) Every term is computable.*

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

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

We have to show that

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

for computable M. So

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

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

(2) `if` is computable:

We have to show that

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

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

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

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

Assume that $[\![\mathtt{cons}_a M]\!] \neq \bot$ for a computable term M of type $\mathtt{I}$. Let $Y \ll [\![\mathtt{cons}_a M]\!] = \widehat{\mathsf{cons}}_a[\![M]\!]$. We need to show that there is i with $Y \sqsubseteq [\mathsf{cons}_a M]_i$. By Lemma 2.4.6, 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 2.4.9 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 $M \xrightarrow{k} \mathtt{cons}_t M'$, $k \leq j$ holds, and so $\mathtt{cons}_a M \xrightarrow{k} \mathtt{cons}_a(\mathtt{cons}_t M') \xrightarrow{1} \mathtt{cons}_{at} M'$. Hence we can take $i = k + 1$.

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

We have to show that if M is computable, then so is $\mathtt{tail}_a M$. Assume that $[\![\mathtt{tail}_a M]\!] \neq \bot$ for a computable term M of type $\mathtt{I}$. Let $Y \ll [\![\mathtt{tail}_a M]\!] = \widehat{\mathsf{tail}}_a[\![M]\!]$. We need to show that there is i with $Y \sqsubseteq [\mathsf{tail}_a M]_i$. By Lemma 2.4.6, there is $X \ll [\![M]\!]$ with $Y \ll \widehat{\mathsf{tail}}_a X$. As M is computable, there is j such that $X \sqsubseteq [M]_j$. It follows that $[M]_j \not\sqsubseteq \{a\}$, because $Y \neq \{\bot\}$ 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 2.4.9 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 $M \xrightarrow{k} \mathtt{cons}_t M', k \leq j$ holds. Because $[M]_j \leq \{a\}$ we conclude that $\mathtt{tail}_a M \xrightarrow{k} \mathtt{tail}_a(\mathtt{cons}_t M') \xrightarrow{1} \mathtt{Ycons}_L$. Hence we can take $i = k + 1$.

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

(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 $Y \sqsubseteq \widehat{\mathsf{tail}}_a[M]_j$. 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 $m \sqsubseteq \mathsf{tail}_a(t) = (a \sqcup t) \setminus a$. 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.

Abstractions

(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 $i = j$.

Fix-point

(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 $\mathtt{Y}_\sigma^{(n)}$ 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 $P \ll [\![\mathtt{Y}^{(n)}N_1 \cdots N_K]\!]$. Since $\mathtt{Y}^{(n)}$ is computable, there is j with $P \sqsubseteq [\mathtt{Y}^{(n)}N_1 \cdots N_k]_j$. Since there is a term M with $\mathtt{Y}^{(n)}N_1 \cdots N_k \xrightarrow{j} \mathtt{cons}_c M$ and from the *syntactic information order* (Definition 5.1.3), which routinely generalizes to our language as Lemma 9.4.8, $\mathtt{Y}^{(n)} \preccurlyeq \mathtt{Y}$ we have that $\mathtt{Y}N_1 \cdots N_k \xrightarrow{j} \mathtt{cons}_c M$ for some M and therefore $i = j$. □

Lemma 9.4.8. *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. $\mathtt{rtest}_{a,b}M \preccurlyeq \mathtt{rtest}_{a,b}M$ is just the definition.

2. $\mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M''$ and $M \to \mathtt{true}$. These follows if

$$\mathtt{rtest}_{a,b}M \to \mathtt{rtest}_{a,b}(\mathtt{cons}_c M''')$$

   and $\overline{c} < b$. By inductive 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. $\mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M''$ and $M \to \mathtt{false}$. Similar to the previous case.

4. $\mathtt{rtest}_{a,b}M' \preccurlyeq \mathtt{rtest}_{a,b}M''$ 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 inductive 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 $\mathtt{true} \preccurlyeq \mathtt{true}, \mathsf{false} \preccurlyeq \mathsf{false}$.

□

9.5 Translation of signed-digit programs to *LRT* programs

In this section, we develop a technique for translating programs from signed-digit representation to programs in the language *LRT*. A similar technique can be used to translate programs from any other concrete representation for exact real-number computation into programs in *LRT*. The main purpose of presenting this translation resides in the use of already existent programs in the literature for exact real number computation.

In Section 4.5, we presented the different approaches to exact real-number computation. Among these approaches, we explained signed-digit representation.

A real number x in signed-digit representation is given by a computable sequence of elements $z = \langle z_0, \ldots, z_i, \ldots \rangle$ over the signed alphabet $3 = \{-1, 0, 1\}$ such that

$$x = \sum_{i \in \mathbb{N}} z_i \times 2^{-i}.$$

Although this representation does not make explicit use of rational (dyadic) intervals, an equivalent representation based on rational (dyadic) intervals

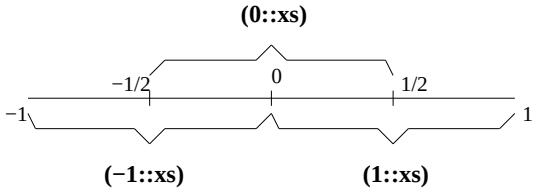


Figure 9.1: Signed-digit Representation.

can be presented. Given a finite part of a sequence which represents a real number x, we can provide an estimation for x with the finite information.

Given a sequence $\langle z_0, \ldots, z_m, \ldots \rangle$ from the element z_m we know that the value of x lies in the interval

$$\left[\left(\sum_{i=1}^{m} z_i \times 2^{-i} \right) - 2^{-m}, \left(\sum_{i=1}^{m} z_i \times 2^{-i} \right) + 2^{-m} \right].$$

Figure 9.1 shows the range of values represented by the signed-digit streams starting with each of the three possible digits.

We use $z_0 :: zs$ instead of $\langle z_0, z_1, z_2, \ldots \rangle$. In this notation z_0 represents an element of the sequence while zs represents a sequence itself, in this case the sequence $\langle z_1, z_2, \ldots \rangle$. This notation is used in most of the functional programming languages when lists of elements are considered. For example, given the sequence $1 :: 0 :: 0 :: 1 :: zs$ the meaning of the four digits presented explicitly amount to saying that the real number represented by this sequence is contained in the interval $[1/2, 5/8]$.

A real number x in the interval representation is obtained as follows:

$$x = \bigcap_{i \in \mathbb{N}} \left[\left(\sum_{j=1}^{i} z_j \cdot 2^{-j} \right) - 2^{-i}, \left(\sum_{j=1}^{i} z_j \cdot 2^{-j} \right) + 2^{-i} \right]$$

Given an input sequence in signed-digit representation, programs are constructed by considering operations over the elements of the input sequence. These operations typically refer to the first element of a sequence, called the head operation, and to the rest of the sequence, called the tail operation.

$$\mathsf{head}(z_0 :: zs) = z_0$$

$$\mathsf{tail}(z_0 :: zs) = zs$$

Most of the time, programmers take advantage of the concrete representation of a sequence in signed-digit representation and they often write

programs analysing a finite number of digits of the input sequences of the program and producing a finite number of digits of the output sequence. Further digits are produced by analysing the remaining digits of the input sequence by recursive calls.

Nevertheless, in our setting we work with an abstract data type for real numbers; hence, programmers do not have access to the representation of real numbers, but they have access to linear maps which can be used to obtain information about the interval in which the solution is contained as in signed-digit representation.

The main goal in this section is to present equivalent primitive constructions in the language *LRT* from those in signed-digit representation. In signed-digit representation programmers have access to each digit of a given sequence, hence they can compare digits explicitly from the representation. However, we are not in the same position. Even more, given two real numbers x, y, we cannot present constructions to check whether $x < y$ or $x = y$ or $x > y$. To overcome this difficulty, the non-deterministic construction in the language *LRT* can be used as explained below.

In order to be able to construct programs which produce the same interval at each stage of a computation as signed-digit representation does, we have to present suitable definitions for the primitive operations which deal with real numbers in the language *LRT*. The three primitive operations for real-number computation in the language *LRT* are $\mathtt{cons}_a, \mathtt{tail}_a$ and $\mathtt{rtest}_{b,c}$.

The operation $\mathtt{cons}_a$ is an increasing affine map which is used to represent real numbers. A real number x is constructed by an infinite composition of the $\mathtt{cons}_a$ operator.

$$x = \mathtt{cons}_{a_1} \circ \mathtt{cons}_{a_2} \circ \cdots \mathtt{cons}_{a_i} \circ \cdots$$

Hence, infinite compositions of the $\mathtt{cons}_a$ operation will be used to represent real numbers in the translation instead of an infinite sequence of digits.

The purpose of the $\mathtt{tail}_a$ operation in *LRT* is as a left inverse operation of $\mathtt{cons}_a$; in the translation it is used as the remainder of an infinite sequence when the leading digit (the digit examined) is removed.

Finally, the $\mathtt{rtest}_{b,c}$ construct will be used to decide whether the real number x is in any of the intervals $[-1, c)$ or $(b, 1]$, with $b < c$. Of course the non-deterministic nature of the construct allows us to present multi-valued programs, but the advantage is that the programs are still sequential.

We are now in a position to present the translation from programs in signed digit representation to programs in *LRT*. From Figure 9.1 it can be seen that infinite sequences of the form $-1 :: zs$, $0 :: zs$ and $1 :: zs$ add up to saying that the real number in question is contained in the intervals $[-1, 0]$, $[-1/2, 1/2]$ and $[0, 1]$ respectively.

Let us define $L = [-1, 0]$, $C = [-1/2, 1/2]$ and $R = [0, 1]$; from Section 3.2.1 we can obtain the following definitions of the maps cons and tail:

$$\mathsf{cons}_L(x) = \frac{x-1}{2}, \quad \mathsf{cons}_C(x) = \frac{x}{2}, \quad \mathsf{cons}_R(x) = \frac{x+1}{2},$$

$$\mathsf{tail}_L(x) = 2x + 1, \quad \mathsf{tail}_C(x) = 2x \text{ and } \quad \mathsf{tail}_R(x) = 2x - 1.$$

9.5.1 Translation

In the translation, we assume that programs for signed-digit representation are defined using conditional expressions rather than pattern matching (see [47, page 17] for a detailed explanation). As an example of a program in the pattern matching style we present the program for the identity function:

$\mathsf{id}(-1 :: xs) = -1 :: \mathsf{id}(xs)$
$\mathsf{id}(0 :: xs) = 0 :: \mathsf{id}(xs)$
$\mathsf{id}(1 :: xs) = 1 :: \mathsf{id}(xs)$

The equivalent definition using conditional expressions is as follows:

```
id(x)= if head(x)=−1
         then  − 1 :: id(tail(x))
         else if head(x)=0
                 then 0 :: id(tail(x))
                 else 1 :: id(tail(x))
```

Given a program P in signed-digit representation, the translation to a program in the language LRT is as follows:

1. Replace -1 by cons_L.
2. Replace 0 by cons_C.
3. Replace 1 by cons_R.
4. Replace :: by $\circ$ (i.e. composition of maps).
5. Replace $\mathsf{tail}(x)$ by $\mathsf{tail}_Y(x)$, where
   - if $x = -1 :: xs$ then $Y = L$,
   - if $x = 0 :: xs$ then $Y = C$,
   - if $x = 1 :: xs$ then $Y = R$.

6. Replace the body of a program of the form:

```
if head(x)=−1
  then A
  else if head(x)=0
         then B
         else C
```

by

```
if rtest_{−1/2,1/2}(x)
  then
       if rtest_{−1/2,0}(x)
         then A
         else B
  else
       if rtest_{0,1/2}(x)
         then B
         else C
```

where $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are subprograms of P.

Hence, as an example, the translation of the program for the identity function given above is as follows:

```
if rtest_{−1/2,1/2}(x)
  then
       if rtest_{−1/2,0}(x)
         then cons_L(id(tail_L(x)))
         else cons_C(id(tail_C(x)))
  else
       if rtest_{0,1/2}(x)
         then cons_C(id(tail_C(x)))
         else cons_R(id(tail_R(x)))
```

Remark 9.5.1. This translation is for algorithms in signed-digit representation which require one digit from the input to be observed in order to produce a digit of the output. A translation for algorithms which consider more than one digit from the input to produce one or more digits as output can be given by a careful, but routine, modification of step 6.

A more elaborated application of this technique is developed in Section 8.5 to derive an *LRT* program for division.

Chapter 10

Correctness of Programs

Computational adequacy allows us to prove partial correctness of programs, but given that denotationally $\perp$ is always a possible result in the Hoare semantics, it does not allow us to prove termination of programs, as explained above. To overcome this problem, we introduce a generalisation of the notion of termination. This notion is based on the idea of convergence of terms. In Section 10.1 we present a definition of strong convergence and some results derived from it. Finally in Section 10.2 we show that the programs which represent the equations presented in Chapter 8 are strongly convergent.

10.1 Termination of Programs

Our main concern is on writing totally correct programs in the programming language *LRT* (see Chapter 9). In order to show that a given program satisfies total correctness with respect to a given specification, we show that

1. If a program converges, it satisfies the specification.
2. That a program in fact converges.

Item 1 will be achieved using the denotational semantics and computational adequacy. Item 2 will be achieved using the operational semantics. Hence, we need to define an operational notion of convergence. However, given the way the operational semantics is defined in Chapter 9.3, it allows divergence when:

- the second rule of $\texttt{cons}_a$ is used infinitely often in a row,
- the fifth rule of $\texttt{tail}_a$ is used infinitely often in a row,
- the third rule of $\texttt{rtest}_{a,b}$ is used infinitely often in a row.

In order to avoid the divergence that arises, we impose a strategy for the application of the rules. Nevertheless, we still keep the rules non-deterministic (the non-determinism is only introduced by $\texttt{rtest}_{a,b}$) and adopt the following particular strategy.

Definition 10.1.1 (Theoretical strategy). *By a theoretical strategy we mean the following:*

- *We never apply the second rule of* $\texttt{cons}_a$*, and the fifth rule of* $\texttt{tail}_a$ *infinitely often unless the other rules are not applicable.*
- *We never apply the third rule for* $\texttt{rtest}_{a,b}$ *infinitely often unless the first two rules are not applicable. Thus, we may still get the two answers* `true` *and* `false` *in some cases for a term of the form* $\texttt{rtest}_{a,b}(M)$ *but we will get a divergent computation for such a term only if no computation leading to* `true` *or* `false` *is possible.*

Thus, in the definition of convergence given below, we quantify over all possible reduction paths except those ruled out by the strategy that we adopt. This theoretical strategy ensures that for any term M which converges to a total real number, $\texttt{rtest}_{a,b}(M)$ always converges to `true` or `false` but never diverges (as specified in Chapter 1).

In a particular implementation of *LRT*, it may be appropriate to further refine the theoretical strategy so that the application of the rules becomes completely deterministic.

Definition 10.1.2 (Practical strategy). *By a practical strategy we mean a strategy such that:*

- *The application of the rules is completely deterministic.*
- *We never apply the third rule for* $\texttt{rtest}_{a,b}$ *infinitely often unless the first two rules are not applicable.*

Examples of practical strategies.

1. In the order that the rules are given in Section 9.3, apply the first one which is applicable.
2. The same strategy as 1, but swap the order of the first two rules for $\texttt{rtest}_{a,b}$.
3. Invoke an oracle to make the decision of which of the first two rules for $\texttt{rtest}_{a,b}$ try to apply first (an oracle is a sequence of binary digits, which is part of the strategy).

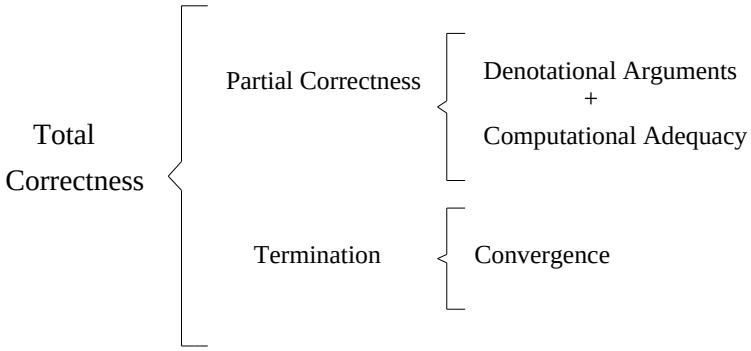


Figure 10.1: Correctness of a program

We now present our definition of strong convergence.

Definition 10.1.3. • *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 $\forall \epsilon > 0, \exists i, \exists M', \exists b$ *such that* $M_i = \mathtt{cons}_b M'$ *and* $\kappa_b < \epsilon$.

- *A term M of type $\sigma \to \tau$ is strongly convergent whenever MN is strongly convergent for every strongly convergent term N of type σ.*

Remark 10.1.4. • Notice that any term which is strongly convergent with respect to the theoretical strategy is also strongly convergent with respect to any practical strategy. Because one cannot be sure what practical strategy is adopted in a particular implementation, we work with the theoretical strategy as this takes into account all possible practical possibilities.

We henceforth refer to strong convergence simply as convergence for the sake of brevity. The following Lemma states that `cons` cannot be reduced to any other term different than itself. This means that we get better or equal partial information in the computation of a term of the form $\mathtt{cons}_a$.

Lemma 10.1.5. *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. According to the operational semantics 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 $b = a$.

□

From Definition 10.1.3 and Lemma 10.1.5 it follows immediately that a term M of type $\mathtt{I}$ is convergent iff for every sequence of reductions

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

and $\forall \epsilon > 0, \exists n, \exists M', \exists a$ such that M_n is of the form $\mathtt{cons}_a M'$ with $\kappa_a < \epsilon$.

If we have a convergent term, all its possible reductions have to be convergent.

Lemma 10.1.6. *If a term M is convergent and $M \to^* N$ then N is convergent.*

Proof. Immediate. □

Our main theorem of this section states that $\mathtt{cons}_a$ and $\mathtt{tail}_a$ are convergent terms. As a corollary of the proof of $\mathtt{cons}_a$ we have that $\mathtt{Ycons}_a$ is convergent.

Theorem 10.1.7. *The terms $\mathtt{cons}_a$ and $\mathtt{tail}_a$ are convergent.*

Proof. We must show that if M is a convergent term then $\mathtt{cons}_a(M)$ and $\mathtt{tail}_a(M)$ are convergent. The proof is by induction over the number of reduction steps. The first is trivial, but the second is not.

1. $\mathtt{cons}_a(M)$ is convergent.

Let

$$\mathtt{cons}_a(M) = M_0 \to^+ M_1 \to^+ M_2 \to^+ \ldots,$$

be an infinite sequence where $+$ stands for one or more reduction steps.

Let $\epsilon > 0$ and put $\delta = \epsilon/\kappa_a$. According to the operational semantics to reduce $\mathtt{cons}_a(M)$ we may apply the rule

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

In order to apply this rule we must reduce M. By hypothesis M is convergent so $\exists N', \exists b$ such that $M \to^* \mathtt{cons}_b N'$ and $\kappa_b < \delta$, then

$$\mathtt{cons}_a(M) \to^+ \mathtt{cons}_a(\mathtt{cons}_b(N')) \to \mathtt{cons}_{ab} N',$$

$$\kappa_{ab} < \kappa_a \cdot \delta = \kappa_a \cdot (\epsilon/\kappa_a) = \epsilon.$$

2. $\mathtt{tail}_a(M)$ is convergent.
Let

$$\mathtt{tail}_a(M) = M_0 \to^+ M_1 \to^+ M_2 \to^+ \dots ,$$

be an infinite reduction sequence.

Let $\epsilon > 0$. According to the operational semantics, to reduce $\mathtt{tail}_a M$ we must apply one of the following rules:

1.$\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathbf{Y}\mathtt{cons}_L$,	if $b \le a$;
2.$\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathbf{Y}\mathtt{cons}_R$,	if $b \ge a$;
3.$\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathtt{cons}_{b\setminus a} M$,	if $a \sqsubset b$ and $a \neq b$;
4.$\mathtt{tail}_a(\mathtt{cons}_b M) \to \mathtt{cons}_{(a\sqcup b)\setminus a}(\mathtt{tail}_{(a\sqcup b)\setminus b} M)$,	if $a \uparrow b, a \not\sqsubseteq b, b \not\sqsubseteq a$, $b \not\le a$ and $a \not\le b$;
5.$\mathtt{tail}_a(M) \to \mathtt{tail}_a(M')$,	if $M \to M'$.

We assume the theoretical strategy presented before. Intuitively this means that rule 5 for tail cannot be applied infinitely often giving rise to a divergent computation, unless rules 1-4 are never applicable.

We can assume that the given reduction starts by n applications of rule 5 by allowing n to be zero. After that we must have an application of one of the rules 1-4 and $\mathtt{tail}_a(M) \to^* \mathtt{tail}_a(\mathtt{cons}_{c_1} M^{(1)})$.

If one of the rules 1,2 or 3 applies then $\mathtt{tail}_a(M)$ is convergent, because the terms $\mathtt{cons}_a, \mathtt{Ycons}_a$ and $M^{(1)}$ are convergent. That $M^{(1)}$ is convergent follows from Lemma 10.1.6. If rule 4 happens we get

$$\mathtt{tail}_a(M) \to^* \mathtt{tail}_a(\mathtt{cons}_{c_1} M^{(1)}) \to \mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1} M^{(1)})$$

where $\alpha_1 = (a \sqcup c_1) \setminus a$ and $\beta_1 = (a \sqcup c_1) \setminus c_1$.

If at this stage $\kappa_{\alpha_1} < \epsilon$ then we have found the required α_1. Otherwise we reduce $\mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1} M^{(1)})$. To reduce this term, we need to reduce $\mathtt{tail}_{\beta_1} M^{(1)}$. Again we assume that the reduction starts by n applications of rule 5 by allowing n to be zero. After that we must have an application of one of the rules 1-4 and $\mathtt{tail}_a(M) \to^* \mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1} M^{(1)}) \to^* \mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_2} M^{(2)}))$. If one of the rules 1, 2 or 3 applies then $\mathtt{tail}_a(M)$ is convergent, because the terms $\mathtt{cons}_a, \mathtt{Ycons}_a$ and $M^{(2)}$ are convergent. If rule 4 follows, we get

$$\mathtt{tail}_a(M) \to^* \mathtt{cons}_{\alpha_1}(\mathtt{cons}_{\alpha_2}(\mathtt{tail}_{\beta_2} M^{(2)}))$$

where $\alpha_2 = (\beta_1 \sqcup c_2) \setminus \beta_1$ and $\beta_2 = (\beta_1 \sqcup c_2) \setminus c_2$. According to the reduction rules for $\mathtt{cons}_a$, we reduce $\mathtt{cons}_{\alpha_1}\mathtt{cons}_{\alpha_2}$ (which is valid according to the operational semantics) and we get $\mathtt{tail}_a(M) \to^* \mathtt{cons}_{\alpha_1\alpha_2}(\mathtt{tail}_{\beta_2}M^{(2)})$ ($\alpha_1\alpha_2$ is an affine map such that $\alpha_1 \sqsubseteq \alpha_1\alpha_2$). If at this stage $\kappa_{\alpha_1\alpha_2} < \epsilon$ then we have found the required interval. Otherwise we carry on reducing $\mathtt{tail}_{\beta_2}M^{(2)}$. We can observe that

$$\begin{array}{rcl} M \to^* \mathtt{cons}_{c_1}M^{(1)} & \to^* & \mathtt{cons}_{c_1}(\mathtt{cons}_{c_2}M^{(2)}) \to^1 \mathtt{cons}_{c_1c_2}M^{(2)} \\ & \to^* & \mathtt{cons}_{c_1c_2}(\mathtt{cons}_{c_3}M^{(3)}) \to^1 \mathtt{cons}_{c_1c_2c_3}M^{(3)} \\ & \to^* & \cdots \ . \end{array}$$

Due to the fact that M is convergent the chain $c_1 \sqsubseteq c_1c_2 \sqsubseteq c_1c_2c_3 \sqsubseteq \cdots$ converges.

If we cannot apply rules 1-3 and we carry on applying rules 5 and 4 respectively, we get:

$$\begin{array}{rcl} \mathtt{tail}(M) & \to^* & \mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1}M^{(1)}) \\ & \to^* & \mathtt{cons}_{\alpha_1}(\mathtt{cons}_{\alpha_2}(\mathtt{tail}_{\beta_2}M^{(2)})) \to^1 \mathtt{cons}_{\alpha_1\alpha_2}(\mathtt{tail}_{\beta_2}M^{(2)}) \\ & \to^* & \mathtt{cons}_{\alpha_1\alpha_2}(\mathtt{cons}_{\alpha_3}(\mathtt{tail}_{\beta_3}M^{(3)})) \to^1 \mathtt{cons}_{\alpha_1\alpha_2\alpha_3}(\mathtt{tail}_{\beta_3}M^{(3)}) \\ & \to^* & \cdots \ . \end{array}$$

where

$$\alpha_1 = (a \sqcup c_1) \setminus a, \qquad \beta_1 = (a \sqcup c_1) \setminus c_1, \tag{10.1}$$

$$\forall n > 1,\ \alpha_n = (\beta_{n-1} \sqcup c_n) \setminus \beta_{n-1}, \qquad \beta_n = (\beta_{n-1} \sqcup c_n) \setminus c_n.$$

This general case is illustrated with the following picture:

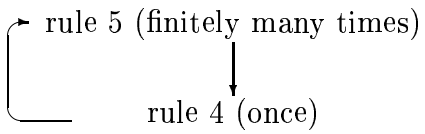


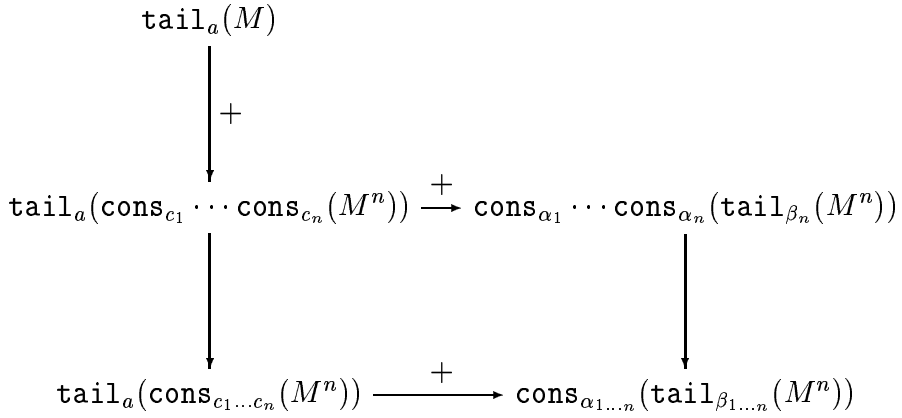
Figure 10.2: Application of rules for $\mathtt{tail}_a$

In order to find an interval whose length is smaller than the given ϵ, we must show that $\kappa_{\prod_n \alpha_n} < \epsilon$. By Lemma 10.1.8 below, for $n = 2$ the same interval is obtained either:

- if M is reduced to $\mathtt{cons}_{c_1}(\mathtt{cons}_{c_2}(M^{(2)}))$ and then $\mathtt{tail}_a(\mathtt{cons}_{c_1c_2}(M^{(2)}))$ is reduced to $\mathtt{cons}_{\alpha_{1,2}}(\mathtt{tail}_{\beta_{1,2}}(M^{(2)}))$, or

- if $\mathtt{tail}_a(\mathtt{cons}_{c_1}(\mathtt{cons}_{c_2} M^{(2)}))$ is reduced to $\mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_2} M^{(2)}))$ and then $\mathtt{cons}_{\alpha_1}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_2} M^{(2)}))$ is reduced to $\mathtt{cons}_{\alpha_1\alpha_2}(\mathtt{tail}_{\beta_2}(M^{(2)}))$.

Using the same reasoning as above, it is immediately clear that for n arbitrary, this still holds as illustrated in the following diagram:



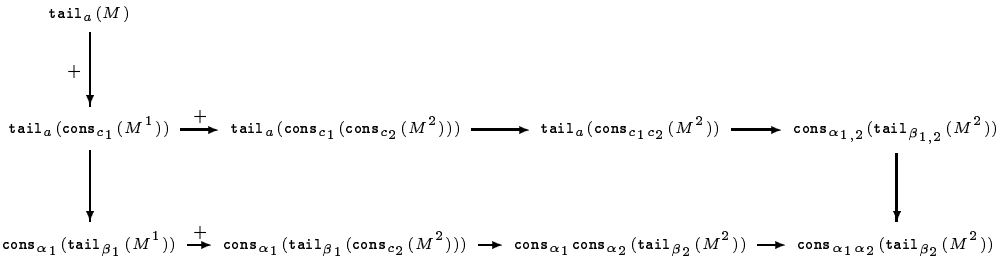
In other words, the same interval is obtained no matter how the reduction rules illustrated in Figure 10.2 are applied.

Hence, without lost of generality, if M is reduced to $\mathtt{cons}_{c_1} \cdots \mathtt{cons}_{c_n}(M^n)$ where $\kappa_{c_1 \cdots c_n} < \frac{\epsilon}{\kappa_a}$ then $\kappa_{\prod_n \alpha_n} < \epsilon$ as required. □

Lemma 10.1.8. *For any term M of the form*

$$M \to \mathtt{cons}_{c_1} M^1 \to \mathtt{cons}_{c_1}\mathtt{cons}_{c_2} M^2,$$

and any non-maximal $a \in \mathcal{I}$, we have the following commuting reductions:



it is to say $\alpha_{1,2} = \alpha_1\alpha_2$ and $\beta_{1,2} = \beta_2$, where the pairs $(a, c_1), (a, c_1c_2)$ and (β_1, c_2) satisfy the conditions of rule 4 for $\mathtt{tail}_a$ and the α_i, β_i are defined in Equation 10.1

Proof. There are two cases to consider. *Case 1.* $a{\uparrow}c_1$ and $\underline{c_1} < \overline{a}$. In this case:

$$\alpha_1\alpha_2 = ((a \sqcup c_1) \setminus a)\, \alpha_2,$$

and $\alpha_2 = \left[\frac{\kappa_{c_1}\underline{c_2}}{\overline{a}-\underline{c_1}}, 1\right]$, hence

$$\alpha_1\alpha_2 = \left[\frac{\underline{c_1}-\underline{a}}{\overline{a}-\underline{a}}, 1\right]\left[\frac{\kappa_{c_1}\underline{c_2}}{\overline{a}-\underline{c_1}}, 1\right] = \left[\left(\frac{\overline{a}-\underline{c_1}}{\overline{a}-\underline{a}}\right)\left(\frac{\kappa_{c_1}\underline{c_2}}{\overline{a}-\underline{c_1}}\right) + \frac{\underline{c_1}-\underline{a}}{\overline{a}-\underline{a}}, 1\right] =$$

$$\left[\frac{\kappa_{c_1}\underline{c_2}+\underline{c_1}-\underline{a}}{\overline{a}-\underline{a}}, 1\right] = \left[\frac{\underline{c_1c_2}-\underline{a}}{\overline{a}-\underline{a}}, 1\right] = [\underline{c_1c_2}, a] \setminus a = (a \sqcup c_1c_2) \setminus a =$$

$$(a \sqcup (c_1c_2)) \setminus a = \alpha_{1,2}.$$

$$\beta_2 = (c_2 \sqcup \beta_1) \setminus c_2 = (c_2 \sqcup ([\underline{c_1}, \overline{a}] \setminus c_1)) \setminus c_2 = \left(c_2 \sqcup \left[0, \frac{\overline{a}-\underline{c_1}}{\kappa_{c_1}}\right]\right) \setminus c_2 =$$

$$\left[\underline{c_2}, \frac{\overline{a}-\underline{c_1}}{\kappa_{c_1}}\right] \setminus c_2 = \left[0, \frac{\overline{a}-\underline{c_1c_2}}{\kappa_{c_1c_2}}\right] = [\underline{c_1c_2}, \overline{a}] \setminus c_1c_2 =$$

$$(c_1c_2 \sqcup a) \setminus c_1c_2 = \beta_{1,2}.$$

The proof of the second case $a{\uparrow}c_1$ and $\underline{a} < \overline{c_1}$ follows similarly. □

Corollary 10.1.9. *The term* $\texttt{Ycons}_a$ *is convergent.*

Proof. Immediate. □

10.2 Convergence of Programs

In this section we show termination of the programs which represent the equations defined in Chapter 8. These proofs together with partial correctness show that our programs are totally correct, in the sense that for any given input, they produce the correct answer (in the case of real number computation, they shrink to the exact solution).

Remark 10.2.1. Although item 2 of Lemmas 8.1.2, 8.2.2, 8.3.2, 8.4.2 and 8.5.2 capture precisely the convergence of the programs presented below, $\perp$ is always a possible value in the Hoare Semantics which in our programs means divergence. Hence, we have to present the proofs of the convergence of the programs when the theoretical strategy is assumed, which as mentioned before, captures all possible practical strategies.

It will be noticed that there are some common parts in the proofs of convergence and their semantics counterpart. The induction over the number of unwindings of the recursive definition in the convergence proofs is equivalent to the induction over the F_ns in their semantics counterpart. However, the proofs are not the same because in the case of convergence, one has to check that a complete unwinding has to happen after finitely many steps while in the semantics this is not assumed. At present it is not clear to us how to relate these proofs in order to make a simplification. This will be explored in further research.

10.2.1 Complement Function

We show termination of the program that implements the complement operation

$$comp(x) = 1 - x,$$

given in section 8.1. We write the following program as a shorthand for a definition in our language using the recursion combinator:

$\texttt{comp}(x)$= if $\texttt{rtest}_{1/3,2/3}(x)$
then
if $\texttt{rtest}_{1/3,1/2}(x)$
then $\texttt{cons}_R(\texttt{comp}(\texttt{tail}_L(x)))$
else $\texttt{cons}_C(\texttt{comp}(\texttt{tail}_C(x)))$
else
if $\texttt{rtest}_{1/2,2/3}(x)$
then $\texttt{cons}_C(\texttt{comp}(\texttt{tail}_C(x)))$
else $\texttt{cons}_L(\texttt{comp}(\texttt{tail}_R(x)))$

where $L = [0, 1/2], C = [1/3, 2/3], R = [1/2, 1]$.

Theorem 10.2.2. *The term* comp *is convergent.*

Proof. We have to show that if $N_1 : \texttt{I}$ is a convergent term then $\texttt{comp}(N_1)$ is convergent. The proof is by induction over the number of unfoldings of the recursive definition. Let

$$\texttt{comp}(N_1) = M_0 \to^+ M_1 \to^+ M_2 \to \cdots,$$

be an infinite reduction sequence. Let $\left(\frac{1}{2}\right)^n < \epsilon, n > 0$. According to the operational semantics, to reduce $\texttt{comp}(N_1)$, we must reduce $\texttt{if}\ \texttt{rtest}_{1/3,2/3}(N_1)$ then we must apply one of the following rules:

i $\texttt{if true}\ M\ N \to M$,

ii $\texttt{if false}\ M\ N \to N$,

iii $\texttt{if}\ B\ M\ N \to\ \texttt{if}\ B'\ M\ N \quad$ if $B \to B'$.

In order to reduce $\texttt{if}\ \texttt{rtest}_{1/3,2/3}(N_1)$ we need to reduce $\texttt{rtest}_{1/3,2/3}(N_1)$.

By assumption, rule 3 of $\texttt{rtest}_{a,b}$ (see section 9.3) cannot be applied infinitely often giving rise to a divergent computation, unless rules 1 and 2 are never applicable, but as N_1 is convergent it never happens. We start by n applications of rule 3, allowing n to be zero. After that, we must have an application of one of the rules 1 or 2 hence we can apply i or ii (at this point $\texttt{rtest}_{1/3,2/3}(N_1) \to^* \texttt{rtest}_{1/3,2/3}(\texttt{cons}_c N_1')$.

We proceed by induction on n. In the base case, there are 3 possibilities:

1. $L \ll c$

$$\texttt{comp}(N_1) \to^* \texttt{cons}_R(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c N_1'))).$$

Take $M_n = \texttt{cons}_R(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c(N_1'))))$ and we have $\kappa_R < \epsilon$.

2. $C \ll c$

$$\texttt{comp}(N_1) \to^* \texttt{cons}_C(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c N_1'))).$$

Take $M_n = \texttt{cons}_C(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c(N_1'))))$ and we have $\kappa_C < \epsilon$.

3. $R \ll c$

$$\texttt{comp}(N_1) \to^* \texttt{cons}_L(\texttt{comp}(\texttt{tail}_R(\texttt{cons}_c N_1'))).$$

Take $M_n = \texttt{cons}_L(\texttt{comp}(\texttt{tail}_R(\texttt{cons}_c(N_1'))))$ and we have $\kappa_L < \epsilon$.

Suppose it is true for n and we want to show that if $(1/2)^{n+1} < \epsilon$, $\exists M_n, \exists b$ such that $M_n = \texttt{cons}_b M'$ and $\kappa_b < \epsilon$.

We know by inductive hypothesis that:

$$\texttt{comp}(N_1) \to^* \texttt{cons}_\alpha(\texttt{comp}(\texttt{tail}_{\beta_1}(\texttt{cons}_{c_n} N_1'))),$$

with $\kappa_\alpha = (1/2)^n < \epsilon'$.

According to the operational semantics, to reduce

$$\texttt{cons}_\alpha(\texttt{comp}(\texttt{tail}_{\beta_1}(\texttt{cons}_{c_n} N_1'))),$$

we need to reduce $\texttt{comp}(\texttt{tail}_{\beta_1}(\texttt{cons}_{c_n} N_1'))$.

By Theorem 10.1.7 $\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1')$ is convergent, then there is c_{n+1} such that

$$\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1') \to^* \mathtt{cons}_{c_{n+1}} N_{n+1}.$$

Hence

$\mathtt{rtest}_{1/3,2/3}(\mathtt{cons}_{c_{n+1}} N_{n+1}) \to \mathtt{true}$ (or to $\mathtt{false}$ depending of c_{n+1}).

Then again we have three cases:

1. $L \ll c_{n+1}$

   $\mathtt{comp}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n))) \to^* \mathtt{cons}_R(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1})))$.

   By the inductive hypothesis and the last reduction:

   $$\mathtt{comp}(N_1) \to^* \mathtt{cons}_\alpha \mathtt{cons}_R(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))),$$

   and $\kappa_{\alpha R} = \kappa_\alpha \kappa_R < \epsilon'(1/2) < \epsilon$.

2. $C \ll c_{n+1}$

   $\mathtt{comp}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n))) \to^* \mathtt{cons}_C(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1})))$.

   By the inductive hypothesis and the last reduction:

   $$\mathtt{comp}(N_1) \to^* \mathtt{cons}_\alpha \mathtt{cons}_C(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))),$$

   and $\kappa_{\alpha C} = \kappa_\alpha \kappa_C < \epsilon'(1/2) < \epsilon$.

3. $R \ll c_{n+1}$. Similar to 1, with $\kappa_{\alpha L} = \kappa_\alpha \kappa_L < \epsilon'(1/2) < \epsilon$.

□

10.2.2 Absolute Value Function

We show termination of the program that implements the absolute value operation

$$abs(x) = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{otherwise;} \end{cases}$$

given in section 8.2. We write the following program as a shorthand for a definition in our language using the recursion combinator:

$$
\begin{array}{l}
\texttt{abs}(x)\texttt{= if rtest}_{-1/3,1/3}(x) \\
\qquad\qquad\texttt{then} \\
\qquad\qquad\quad\texttt{if rtest}_{-1/3,0}(x) \\
\qquad\qquad\qquad\texttt{then cons}_R(\texttt{comp}(\texttt{tail}_L(x))) \\
\qquad\qquad\qquad\texttt{else cons}_C(\texttt{abs}(\texttt{tail}_C(x))) \\
\qquad\qquad\texttt{else} \\
\qquad\qquad\quad\texttt{if rtest}_{0,1/3}(x) \\
\qquad\qquad\qquad\texttt{then cons}_C(\texttt{abs}(\texttt{tail}_C(x))) \\
\qquad\qquad\qquad\texttt{else cons}_R(\texttt{abs}(\texttt{tail}_R(x)))
\end{array}
$$

where $L = [-1/2, 0], C = [-1/3, 1/3], R = [0, 1/2]$.

Theorem 10.2.3. *The term* `abs` *is convergent.*

Proof. We have to show that if $N_1 : \texttt{I}$ is a convergent term then $\texttt{abs}(N_1)$ is convergent. The proof is by induction over the number of unfoldings of the recursive definition. Let

$$\texttt{abs}(N_1) = M_0 \to^+ M_1 \to^+ M_2 \to \cdots,$$

be an infinite reduction sequence. Let $\left(\frac{1}{2}\right)^n < \epsilon, n > 0$. According to the operational semantics, to reduce $\texttt{abs}(N_1)$, we must reduce $\texttt{if rtest}_{-1/3,1/3}(N_1)$ then we must apply one of the following rules:

i $\texttt{if true}\ M\ N \to M$,

ii $\texttt{if false}\ M\ N \to N$,

iii $\texttt{if}\ B\ M\ N \to\ \texttt{if}\ B'\ M\ N \quad$ if $B \to B'$.

In order to reduce $\texttt{if rtest}_{-1/3,1/3}(N_1)$ we need to reduce $\texttt{rtest}_{-1/3,1/3}(N_1)$.

By assumption, rule 3 of $\texttt{rtest}_{a,b}$ (see section 9.3) cannot be applied infinitely many often giving rise to a divergent computation, unless rules 1 and 2 are never applicable, but as N_1 is convergent it never happens. We start by n applications of rule 3, allowing n to be zero. After that, we must have an application of one of the rules 1 or 2 hence we can apply i or ii (at this point $\texttt{rtest}_{-1/3,1/3}(N_1) \to^* \texttt{rtest}_{-1/3,1/3}(\texttt{cons}_c N_1')$.

We proceed by induction on n. In the base case, there are 3 possibilities:

1. $L \ll c$, then:

$$\texttt{abs}(N_1) \to^* \texttt{cons}_R(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c N_1'))).$$

Take $M_n = \texttt{cons}_R(\texttt{comp}(\texttt{tail}_L(\texttt{cons}_c(N_1'))))$ and we have $\kappa_R < \epsilon$.

2. $C \ll c$

$$\mathtt{abs}(N_1) \to^* \mathtt{cons}_C(\mathtt{abs}(\mathtt{tail}_C(\mathtt{cons}_c N_1^{'}))).$$

Take $M_n = \mathtt{cons}_C(\mathtt{abs}(\mathtt{tail}_C(\mathtt{cons}_c(N_1'))))$ and we have $\kappa_C < \epsilon$.

3. $R \ll c$

$$\mathtt{abs}(N_1) \to^* \mathtt{cons}_R(\mathtt{abs}(\mathtt{tail}_R(\mathtt{cons}_c N_1^{'}))).$$

Take $M_n = \mathtt{cons}_R(\mathtt{abs}(\mathtt{tail}_R(\mathtt{cons}_c(N_1'))))$ and we have $\kappa_R < \epsilon$.

Suppose it is true for n and we want to show that if $(1/2)^{n+1} < \epsilon$, $\exists M_n, \exists b$ such that $M_n = \mathtt{cons}_b M'$ and $\kappa_b < \epsilon$.

We know by inductive hypothesis that:

$$\mathtt{abs}(N_1) \to^* \mathtt{cons}_\alpha(\mathtt{comp}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}))),$$

or

$$\mathtt{abs}(N_1) \to^* \mathtt{cons}_\alpha(\mathtt{abs}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}))),$$

with $\kappa_\alpha = (1/2)^n < \epsilon'$.

In the first case, as the complement map is convergent 10.2.1, there is nothing to do.

In the second case, according to the operational semantics, to reduce

$$\mathtt{cons}_\alpha(\mathtt{abs}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}))),$$

we need to reduce $\mathtt{abs}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}))$.

By Theorem 10.1.7 $\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'})$ is convergent, then there is c_{n+1} such that

$$\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}) \to^* \mathtt{cons}_{c_{n+1}} N_{n+1}.$$

Hence

$\mathtt{rtest}_{-1/3,1/3}(\mathtt{cons}_{c_{n+1}} N_{n+1}) \to \mathtt{true}$ (or to $\mathtt{false}$ depending of c_{n+1}).

Then again we have three cases:

1. $L \ll c_{n+1}$, hence

$$\mathtt{abs}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n))) \to^* \mathtt{cons}_R(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))).$$

By the inductive hypothesis and the last reduction:

$$\mathtt{abs}(N_1) \to^* \mathtt{cons}_\alpha \mathtt{cons}_R(\mathtt{comp}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))),$$

and $\kappa_{\alpha R} = \kappa_\alpha \kappa_R < \epsilon'(1/2) < \epsilon$.

2. $C \ll c_{n+1}$ hence

   $$\mathtt{abs}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n))) \to^* \mathtt{cons}_C(\mathtt{abs}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))).$$

   By the inductive hypothesis and the last reduction:

   $$\mathtt{abs}(N_1) \to^* \mathtt{cons}_\alpha \mathtt{cons}_C(\mathtt{abs}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}))),$$

   and $\kappa_{\alpha C} = \kappa_\alpha \kappa_C < \epsilon'(1/2) < \epsilon$.

3. $R \ll c_{n+1}$. Similar to 1.

□

10.2.3 Average Function

We show termination of a recursive program that implements the average operation

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

defined in Section 8.3. We write the following program as a shorthand for a definition in our language using the recursion combinator:

```
Average(x1, x2)= if rtest_{0,3/4}(x1)
                 then
                     if rtest_{0,3/4}(x2)
                        then cons_L(Average(tail_L(x1), tail_L(x2)))
                        else cons_C(Average(tail_L(x1), tail_R(x2)))
                 else
                     if rtest_{0,3/4}(x2)
                        then cons_C(Average(tail_R(x1), tail_L(x2)))
                        else cons_R(Average(tail_R(x1), tail_R(x2))),
```

where $L = [0, 3/4], C = [1/8, 7/8], R = [1/4, 1]$.

Theorem 10.2.4. *The term* `Average` *is convergent.*

Proof. We have to show that if $N_1, N_2 : \mathtt{I}$ are convergent terms then $\mathtt{Average}(N_1, N_2)$ is convergent. The proof is by induction over the number of unfoldings of the recursive definition. Let

$$\mathtt{Average}(N_1, N_2) = M_0 \to^+ M_1 \to^+ M_2 \to \cdots,$$

be an infinite reduction sequence. Let $\left(\frac{3}{4}\right)^n < \epsilon, n > 0$. According to the operational semantics, to reduce $\mathtt{Average}(N_1, N_2)$, we must reduce $\mathtt{if}\ \mathtt{rtest}_{0,3/4}(N_1)$ then we must apply one of the following rules:

i $\texttt{if true}\ M\ N \to M$,

ii $\texttt{if false}\ M\ N \to N$,

iii $\texttt{if}\ B\ M\ N \to\ \texttt{if}\ B'\ M\ N \quad$ if $B \to B'$.

In order to reduce $\texttt{if}\ \texttt{rtest}_{0,3/4}(N_1)$ we need to reduce $\texttt{rtest}_{0,3/4}(N_1)$.

By assumption, rule 3 of $\texttt{rtest}_{a,b}$ (see section 9.3) cannot be applied infinitely many often giving rise to a divergent computation, unless rules 1 and 2 are never applicable, but as N_1 is convergent it never happens. We start by n applications of rule 3, allowing n to be zero. After that, we must have an application of one of the rules 1 or 2 hence we can apply i or ii (at this point $\texttt{rtest}_{0,3/4}(N_1) \to^* \texttt{rtest}_{0,3/4}(\texttt{cons}_c N_1')$.

The same argument as above follows for reducing $\texttt{if}\ \texttt{rtest}_{0,3/4}(N_2)$ and $\texttt{rtest}_{0,3/4}(N_2) \to^* \texttt{rtest}_{0,3/4}(\texttt{cons}_d N_2')$. We proceed by induction on n. In the base case, there are 4 possibilities:

1. $L \ll c, L \ll d$

   $$\texttt{Average}(N_1, N_2) \to^* \texttt{cons}_L(\texttt{Average}(\texttt{tail}_L(\texttt{cons}_c N_1^{'}), \texttt{tail}_L(\texttt{cons}_d N_2^{'}))).$$

   Take $M_n = \texttt{cons}_L(\texttt{Average}(\texttt{tail}_L(\texttt{cons}_c(N_1')), \texttt{tail}_L(\texttt{cons}_d(N_2'))))$ and we have $\kappa_L < \epsilon$.

2. $L \ll c, R \ll d$

   $$\texttt{Average}(N_1, N_2) \to^* \texttt{cons}_C(\texttt{Average}(\texttt{tail}_L(\texttt{cons}_c N_1^{'}), \texttt{tail}_R(\texttt{cons}_d N_2^{'}))).$$

   Take $M_n = \texttt{cons}_C(\texttt{Average}(\texttt{tail}_L(\texttt{cons}_c(N_1')), \texttt{tail}_R(\texttt{cons}_d(N_2'))))$ and we have $\kappa_C < \epsilon$.

3. $R \ll c, L \ll d$ Equivalent to 2.

4. $R \ll c, R \ll d$

   $$\texttt{Average}(N_1, N_2) \to^* \texttt{cons}_R(\texttt{Average}(\texttt{tail}_R(\texttt{cons}_c N_1^{'}), \texttt{tail}_R(\texttt{cons}_d N_2^{'}))).$$

   Take $M_n = \texttt{cons}_R(\texttt{Average}(\texttt{tail}_R(\texttt{cons}_c(N_1')), \texttt{tail}_R(\texttt{cons}_d(N_2'))))$ and we have $\kappa_R < \epsilon$.

Suppose it is true for n and we want to show that if $(3/4)^{n+1} < \epsilon$, $\exists M_n, \exists b$ such that $M_n = \mathtt{cons}_b M'$ and $\kappa_b < \epsilon$.

We know by inductive hypothesis that:

$$\mathtt{Average}(N_1, N_2) \to^* \mathtt{cons}_\alpha(\mathtt{Average}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'}))),$$

with $\kappa_\alpha = (3/4)^n < \epsilon'$.

According to the operational semantics, to reduce

$$\mathtt{cons}_\alpha(\mathtt{Average}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'}))),$$

we need to reduce $\mathtt{Average}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'}))$.

By Theorem 10.1.7 $\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'})$ and $\mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'})$ are convergent, then there are c_{n+1}, d_{n+1} such that

$$\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}) \to^* \mathtt{cons}_{c_{n+1}} N_{n+1}.$$

and

$$\mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_n^{'}) \to^* \mathtt{cons}_{d_{n+1}} N'_{n+1}.$$

Hence

$$\mathtt{rtest}_{0,3/4}(\mathtt{cons}_{c_{n+1}} N_{n+1}) \to \mathtt{true} \text{ (or to } \mathtt{false} \text{ depending of } c_{n+1}),$$

and

$$\mathtt{rtest}_{0,3/4}(\mathtt{cons}_{d+1} N'_{n+1}) \to \mathtt{true} \text{ (or to } \mathtt{false} \text{ depending of } d_{n+1}).$$

Then again we have four cases:

1. $L \ll c_{n+1} \& L \ll d_{n+1}$

   $$\mathtt{Average}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n)), \mathtt{tail}_{\beta'_n}(\mathtt{cons}_{d_n}(N_n^{'}))) \to^* \mathtt{cons}_L(\mathtt{Average}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N_{n+1}^{'}))).$$

   By the inductive hypothesis and the last reduction:

   $$\mathtt{Average}(N_1, N_2) \to^* \mathtt{cons}_\alpha \mathtt{cons}_L(\mathtt{Average}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N_{n+1}^{'}))),$$

   and $\kappa_{\alpha L} = \kappa_\alpha \kappa_L < \epsilon'(3/4) < \epsilon$.

2. $L \ll c_{n+1}$ & $R \ll d_{n+1}$

$$\begin{aligned}&\mathtt{Average}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n)), \mathtt{tail}_{\beta'_n}(\mathtt{cons}_{d_n}(N'_n))) \rightarrow^* \\ &\qquad \mathtt{cons}_C(\mathtt{Average}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \\ &\qquad \mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))).\end{aligned}$$

By the inductive hypothesis and the last reduction:

$$\begin{aligned}&\mathtt{Average}(N_1, N_2) \rightarrow^* \\ &\qquad \mathtt{cons}_\alpha \mathtt{cons}_C(\mathtt{Average}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \\ &\qquad \mathtt{tail}'_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))),\end{aligned}$$

and $\kappa_{\alpha C} = \kappa_\alpha \kappa_C < \epsilon'(3/4) < \epsilon$.

3. $R \ll c_{n+1}$ & $L \ll d_{n+1}$. Similar to 2.

4. $R \ll c_{n+1}$ & $R \ll d_{n+1}$. Similar to 1, with $\kappa_{\alpha R} = \kappa_\alpha \kappa_R < \epsilon'(3/4) < \epsilon$.

□

10.2.4 Multiplication Function

We show termination of a recursive program that implements the multiplication operation

$$mult(x, y) = x \times y.$$

defined in Section 8.4. We write the following program as a shorthand for a definition in our language using the recursion combinator:

$$\begin{aligned}&\mathtt{mult}(X, Y) = \\ &\quad \mathtt{if}\ \mathtt{rtest}_{0,2/3}(X) \\ &\qquad \mathtt{then} \\ &\qquad\quad \mathtt{if}\ \mathtt{rtest}_{0,2/3}(Y) \\ &\qquad\qquad \mathtt{then}\ \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_L(X), \mathtt{tail}_L(Y))) \\ &\qquad\qquad \mathtt{else}\ \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_L(X), \mathtt{tail}_R(Y))) + \tfrac{2}{9}(\mathtt{tail}_L(X)) \\ &\qquad \mathtt{else} \\ &\qquad\quad \mathtt{if}\ \mathtt{rtest}_{0,2/3}(Y) \\ &\qquad\qquad \mathtt{then}\ \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_R(X), \mathtt{tail}_L(Y))) + \tfrac{2}{9}(\mathtt{tail}_L(Y)) \\ &\qquad\qquad \mathtt{else}\ \mathtt{cons}_{C_2}(\mathtt{mult}(\mathtt{tail}_R(X), \mathtt{tail}_R(Y))) + \\ &\qquad\qquad\qquad\qquad \tfrac{2}{9}(\mathtt{tail}_R(X) + \mathtt{tail}_R(Y)),\end{aligned}$$

where $L = [0, 2/3], R = [1/3, 1], C_1 = [0, 4/9]$ and $C_2 = [1/9, 5/9]$.

Theorem 10.2.5. *The term* `mult` *is convergent.*

Proof. We have to show that if $N_1, N_2 : \mathtt{I}$ are convergent terms then $\mathtt{mult}(N_1, N_2)$ is convergent. The proof is by induction over the number of unfoldings of the recursive definition. Let

$$\mathtt{mult}(N_1, N_2) = M_0 \to^+ M_1 \to^+ M_2 \to \cdots,$$

be an infinite reduction sequence. Let $\left(\frac{3}{4}\right)^n < \epsilon, n > 0$. According to the operational semantics, to reduce $\mathtt{mult}(N_1, N_2)$, we must reduce $\mathtt{if}\ \mathtt{rtest}_{0,2/3}(N_1)$ then we must apply one of the following rules:

i $\mathtt{if}\ \mathtt{true}\ M\ N \to M$,

ii $\mathtt{if}\ \mathtt{false}\ M\ N \to N$,

iii $\mathtt{if}\ B\ M\ N \to\ \mathtt{if}\ B'\ M\ N \quad$ if $B \to B'$.

In order to reduce $\mathtt{if}\ \mathtt{rtest}_{0,2/3}(N_1)$ we need to reduce $\mathtt{rtest}_{0,2/3}(N_1)$.

By assumption, rule 3 of $\mathtt{rtest}_{a,b}$ (see section 9.3) cannot be applied infinitely many often giving rise to a divergent computation, unless rules 1 and 2 are never applicable, but as N_1 is convergent it never happens. We start by n applications of rule 3, allowing n to be zero. After that, we must have an application of one of the rules 1 or 2 hence we can apply i or ii (at this point $\mathtt{rtest}_{0,2/3}(N_1) \to^* \mathtt{rtest}_{0,2/3}(\mathtt{cons}_c N_1')$.

The same argument as above follows for reducing $\mathtt{if}\ \mathtt{rtest}_{0,2/3}(N_2)$ and $\mathtt{rtest}_{0,2/3}(N_2) \to^* \mathtt{rtest}_{0,2/3}(\mathtt{cons}_d N_2')$. We proceed by induction on n. In the base case, there are 4 possibilities:

1. $L \ll c, L \ll d$

   $$\mathtt{mult}(N_1, N_2) \to^* \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_L(\mathtt{cons}_c N_1^{'}), \mathtt{tail}_L(\mathtt{cons}_d N_2^{'}))).$$

   Take $M_n = \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_L(\mathtt{cons}_c(N_1')), \mathtt{tail}_L(\mathtt{cons}_d(N_2'))))$ and we have $\kappa_{C_1} < \epsilon$.

2. $L \ll c, R \ll d$

   $$\mathtt{mult}(N_1, N_2) \to^* \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_L(\mathtt{cons}_c N_1^{'}), \mathtt{tail}_R(\mathtt{cons}_d N_2^{'}))) + \frac{2}{9}\left(\mathtt{tail}_L(\mathtt{cons}_c N_1^{'})\right).$$

   Take $M_n = \mathtt{cons}_{C_1}(\mathtt{Average}(\mathtt{tail}_L(\mathtt{cons}_c(N_1')), \mathtt{tail}_R(\mathtt{cons}_d(N_2'))))$ $+\frac{2}{9}\left(\mathtt{tail}_L(\mathtt{cons}_c N_1^{'})\right)$ and we have $\kappa_{C_1} < \epsilon$ (the $+$ operation is a translation map which does not affect the length).

3. $R \ll c, L \ll d$ Equivalent to 2.

4. $R \ll c, R \ll d$

$$\begin{aligned}\mathtt{mult}(N_1, N_2) \rightarrow^* \mathtt{cons}_{C_2}(\mathtt{mult}(\mathtt{tail}_R(\mathtt{cons}_c N_1^{'}), \mathtt{tail}_R(\mathtt{cons}_d N_2^{'})))\\ + \frac{2}{9}(\mathtt{tail}_R(\mathtt{cons}_c N_1^{'}), \mathtt{tail}_R(\mathtt{cons}_d N_2^{'})).\end{aligned}$$

Take $M_n = \mathtt{cons}_{C_2}(\mathtt{mult}(\mathtt{tail}_R(\mathtt{cons}_c(N_1')), \mathtt{tail}_R(\mathtt{cons}_d(N_2'))))$
$+\frac{2}{9}(\mathtt{tail}_R(\mathtt{cons}_c(N_1')), \mathtt{tail}_R(\mathtt{cons}_d(N_2')))$ and we have $\kappa_{C_2} < \epsilon$.

Suppose it is true for n and we want to show that if $(3/4)^{n+1} < \epsilon$, $\exists M_n, \exists b$ such that $M_n = \mathtt{cons}_b M'$ and $\kappa_b < \epsilon$.

We know by inductive hypothesis that:

$$\begin{aligned}\mathtt{mult}(N_1, N_2) \rightarrow^* \mathtt{cons}_\alpha(\mathtt{mult}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'})))\\ + K,\end{aligned}$$

with $\kappa_\alpha = (3/4)^n < \epsilon'$ and $K < \frac{2}{9}$.

According to the operational semantics, to reduce

$$\mathtt{cons}_\alpha(\mathtt{mult}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'}))) + K,$$

we need to reduce $\mathtt{mult}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}), \mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'}))$.

By Theorem 10.1.7 $\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'})$ and $\mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_2^{'})$ are convergent, then there are c_{n+1}, d_{n+1} such that

$$\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'}) \rightarrow^* \mathtt{cons}_{c_{n+1}} N_{n+1}.$$

and

$$\mathtt{tail}_{\beta_2}(\mathtt{cons}_{d_n} N_n^{'}) \rightarrow^* \mathtt{cons}_{d_{n+1}} N'_{n+1}.$$

Hence

$$\mathtt{rtest}_{0,3/4}(\mathtt{cons}_{c_{n+1}} N_{n+1}) \rightarrow \mathtt{true} \text{ (or to } \mathtt{false} \text{ depending of } c_{n+1}),$$

and

$$\mathtt{rtest}_{0,3/4}(\mathtt{cons}_{d_{n+1}} N'_{n+1}) \rightarrow \mathtt{true} \text{ (or to } \mathtt{false} \text{ depending of } d_{n+1}).$$

Then again we have four cases:

1. $L \ll c_{n+1} \& L \ll d_{n+1}$

$$\begin{aligned}&\mathtt{mult}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n)), \mathtt{tail}_{\beta'_n}(\mathtt{cons}_{d_n}(N'_n))) \to^* \\ &\mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))).\end{aligned}$$

   By the inductive hypothesis and the last reduction:

$$\begin{aligned}\mathtt{mult}(N_1, N_2) \to^* &\mathtt{cons}_\alpha \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \\ &\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))),\end{aligned}$$

   and $\kappa_{\alpha C_1} = \kappa_\alpha \kappa_{C_1} < \epsilon'(3/4) < \epsilon$.

2. $L \ll c_{n+1}$ & $R \ll d_{n+1}$

$$\begin{aligned}&\mathtt{mult}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n)), \mathtt{tail}_{\beta'_n}(\mathtt{cons}_{d_n}(N'_n))) \to^* \\ &\mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))) \\ &+ \frac{2}{9}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1})).\end{aligned}$$

   By the inductive hypothesis and the last reduction:

$$\begin{aligned}\mathtt{mult}(N_1, N_2) \to^* &\ \mathtt{cons}_\alpha \mathtt{cons}_{C_1}(\mathtt{mult}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1}), \\ &\mathtt{tail}'_{\beta_{n+1}}(\mathtt{cons}_{d_{n+1}} N'_{n+1}))) + \frac{2}{9}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}} N_{n+1})),\end{aligned}$$

   and $\kappa_{\alpha C_1} = \kappa_\alpha \kappa_{C_1} < \epsilon'(3/4) < \epsilon$.

3. $R \ll c_{n+1}$ & $L \ll d_{n+1}$. Similar to 2.

4. $R \ll c_{n+1}$ & $R \ll d_{n+1}$. Similar to 1, with $\kappa_{\alpha C_2} = \kappa_\alpha \kappa_{C_2} < \epsilon'(3/4) < \epsilon$.

□

10.2.5 Division Function

We show termination of a recursive program that implements the division operation

$$\mathtt{div}(x, y) = x/4y.$$

defined in Section 8.5. We write the following program as a shorthand for a definition in our language using the recursion combinator:

$$
\begin{array}{l}
\texttt{div}(x,y) = \\
\quad \texttt{if rtest}_{-1/2,1/2}(X) \\
\quad\quad \texttt{then} \\
\quad\quad\quad \texttt{if rtest}_{-1/2,-1/4}(x) \\
\quad\quad\quad\quad \texttt{then} \\
\quad\quad\quad\quad\quad \texttt{if rtest}_{-1/2,1/2}(x+y) \\
\quad\quad\quad\quad\quad\quad \texttt{then} \\
\quad\quad\quad\quad\quad\quad\quad \texttt{if rtest}_{-1/2,-1/4}(x+y) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{then cons}_L(\texttt{div}(\texttt{tail}_C(x+2y),y)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{else cons}_{C_3}(\texttt{div}(\texttt{tail}_C(x+y),y)) \\
\quad\quad\quad\quad\quad\quad \texttt{else} \\
\quad\quad\quad\quad\quad\quad\quad \texttt{if rtest}_{1/4,1/2}(x+y) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{then cons}_{C_3}(\texttt{div}(\texttt{tail}_C(x+y),y)) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{else cons}_{C_4}(\texttt{div}(\texttt{tail}_C(x)+y,y)) \\
\quad\quad\quad\quad \texttt{else cons}_C(\texttt{div}(\texttt{tail}_C(x),y)) \\
\quad\quad \texttt{else} \\
\quad\quad\quad \texttt{if rtest}_{1/4,1/2}(x) \\
\quad\quad\quad\quad \texttt{then cons}_C(\texttt{div}(\texttt{tail}_C(x),y)) \\
\quad\quad\quad\quad \texttt{else} \\
\quad\quad\quad\quad\quad \texttt{if rtest}_{-1/2,1/2}(x-y) \\
\quad\quad\quad\quad\quad\quad \texttt{then} \\
\quad\quad\quad\quad\quad\quad\quad \texttt{if rtest}_{-1/2,-1/4}(x-y) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{then cons}_{C_2}(\texttt{div}(\texttt{tail}_C(x)-y,y)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{else cons}_{C_1}(\texttt{div}(\texttt{tail}_C(x-y),y)) \\
\quad\quad\quad\quad\quad\quad \texttt{else} \\
\quad\quad\quad\quad\quad\quad\quad \texttt{if rtest}_{1/4,1/2}(x-y) \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{then cons}_{C_1}(\texttt{div}(\texttt{tail}_C(x-y),y)) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{else cons}_R(\texttt{div}(\texttt{tail}_C(x-2y),y)),
\end{array}
$$

where $L = [-1,0], C = [-1/2,1/2], R = [0,1], C_1 = [-1/4,3/4], C_2 = [-3/8,5/8], C_3 = [-3/4,1/4]$ and $C_4 = [-5/8,3/8]$.

Theorem 10.2.6. *The term* `div` *is convergent.*

Proof. We have to show that if N_1, N_2 are convergent terms then $\texttt{div}(N_1,N_2)$ is convergent. The proof is by induction over the number of unfoldings of the recursive definition. Let

$$\texttt{div}(N_1,N_2) = M_0 \to^+ M_1 \to^+ M_2 \to \cdots,$$

be an infinite reduction sequence. Let $\left(\frac{1}{2}\right)^{n-2} < \epsilon, n > 0$. According to the operational semantics, to reduce $\texttt{div}(N_1,N_2)$, we must reduce $\texttt{if rtest}_{-1/2,1/2}(N_1)$ then we must apply one of the following rules:

i $\texttt{if}\ \texttt{true}\ M\ N \to M$,

ii $\texttt{if}\ \texttt{false}\ M\ N \to N$,

iii $\texttt{if}\ B\ M\ N \to\ \texttt{if}\ B'\ M\ N \quad$ if $B \to B'$.

In order to reduce $\texttt{if}\ \texttt{rtest}_{-1/2,1/2}(N_1)$ we need to reduce $\texttt{rtest}_{-1/2,1/2}(N_1)$.

By assumption, rule 3 of $\texttt{rtest}_{a,b}$ (see section 9.3) cannot be applied infinitely many often giving rise to a divergent computation, unless rules 1 and 2 are never applicable, but as N_1 is convergent it never happens. We start by n applications of rule 3, allowing n to be zero. After that, we must have an application of one of the rules 1 or 2 hence we can apply i or ii (at this point $\texttt{rtest}_{-1/2,1/2}(N_1) \to^* \texttt{rtest}_{-1/2,1/2}(\texttt{cons}_c N_1')$).

The same argument as above follows for reducing the other conditions. We proceed by induction on n. In the base case, there are 10 possibilities:

1. If $[-1,-1/2] \sqsubseteq c$ and $[-1,-1/2] \sqsubseteq d$ where:

   $$\texttt{rtest}_{-1/2,-1/4}(N_1) \to^* \texttt{rtest}_{-1/2,-1/4}(\texttt{cons}_c N_1')$$

   and

   $$\texttt{rtest}_{-1/2,-1/4}(N_1 + N_2) \to^* \texttt{rtest}_{-1/2,-1/4}(\texttt{cons}_d N_2')$$

   then

   $$\texttt{div}(N_1, N_2) \to^* \texttt{cons}_L(\texttt{div}(\texttt{tail}_C(\texttt{cons}_c N_1^{'} + 2N_2), N_2))$$

   Take $M_n = \texttt{cons}_L(\texttt{div}(\texttt{tail}_C(\texttt{cons}_c(N_1') + 2N_2), N_2'))$ and we have $\kappa_L < \epsilon$.

2. If $[1/2,1] \sqsubseteq c$ and $[1/2,1] \sqsubseteq d$ where:

   $$\texttt{rtest}_{1/4,1/2}(N_1) \to^* \texttt{rtest}_{1/4,1/2}(\texttt{cons}_c N_1')$$

   and

   $$\texttt{rtest}_{1/4,1/2}(N_1 - N_2) \to^* \texttt{rtest}_{1/4,1/2}(\texttt{cons}_d N_2')$$

   then

   $$\texttt{div}(N_1, N_2) \to^* \texttt{cons}_R(\texttt{div}(\texttt{tail}_C(\texttt{cons}_c N_1^{'} - 2N_2), N_2))$$

   Take $M_n = \texttt{cons}_R(\texttt{div}(\texttt{tail}_C(\texttt{cons}_c(N_1') - 2N_2), N_2'))$ and we have $\kappa_R < \epsilon$.

A similar, but routine, analysis follows on the other basic cases.

Suppose it is true for n. We want to show that if $(\frac{1}{2})^{n-1} \leq \epsilon$, then $\exists M_n, \exists b$ such that $M_n = \mathtt{cons}_b M'$ and $\kappa_b < \epsilon$. We know by inductive hypothesis that:

$$\mathtt{div}(N_1, N_2) \rightarrow^* \mathtt{cons}_\alpha(\mathtt{div}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'} + 2N_2), N_2))$$

with $\kappa_\alpha = (1/2)^{n-2} < \epsilon'$.

According to the operational semantics, to reduce

$$\mathtt{cons}_\alpha(\mathtt{div}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'} + 2N_2), N_2)),$$

we need to reduce $\mathtt{div}(\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'} + 2N_2), N_2)$.

By Theorem 10.1.7 $\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'} + 2N_2)$ is convergent, then there is c_{n+1} such that

$$\mathtt{tail}_{\beta_1}(\mathtt{cons}_{c_n} N_1^{'} + 2N_2) \rightarrow^* \mathtt{cons}_{c_{n+1}} N_{n+1}.$$

Hence

$$\mathtt{rtest}_{-1/2,1/2}(\mathtt{cons}_{c_{n+1}} N_{n+1}) \rightarrow \mathtt{true} \text{ (or to } \mathtt{false} \text{ depending of } c_{n+1}),$$

Then again we have ten cases:

1. $[-1, -1/2] \sqsubseteq c_{n+1} \& [-1, -1/2] \sqsubseteq d_n$

   $$\mathtt{div}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n) + 2N_2), N_2) \rightarrow^* \mathtt{cons}_L(\mathtt{div}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}}(N_{n+1}) + 2N_2), N_2)).$$

   By the inductive hypothesis and the last reduction:

   $$\mathtt{div}(N_1, N_2) \rightarrow^* \mathtt{cons}_\alpha \mathtt{cons}_L(\mathtt{div}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}}(N_{n+1}) + 2N_2), N_2)),$$

   and $\mathtt{cons}_\alpha \mathtt{cons}_L = \mathtt{cons}_\alpha([-1, 0])$, and

   $$\begin{aligned} \mathtt{cons}_\alpha([-1, 0]) &= \left[\frac{\kappa_\alpha}{2}(-1) + \frac{\kappa_\alpha}{2}, \frac{\kappa_\alpha}{2}(0) + \frac{\kappa_\alpha}{2}\right] \\ &= \left[0, \frac{\kappa_\alpha}{2}\right], \end{aligned}$$

   hence $\kappa_{\left[0, \frac{|\alpha|}{2}\right]} = \frac{\kappa_\alpha}{2} \leq \epsilon$.

2. $[1/2, 1] \sqsubseteq c_{n+1}$ & $[1/2, 1] \sqsubseteq d_n$

   $$\mathtt{div}(\mathtt{tail}_{\beta_n}(\mathtt{cons}_{c_n}(N_n) + 2N_2), N_2) \rightarrow^* \mathtt{cons}_R(\mathtt{div}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}}(N_{n+1}) - 2N_2), N_2)).$$

By the inductive hypothesis and the last reduction:

$$\begin{aligned}&\mathtt{div}(N_1, N_2) \to^* \\ &\quad \mathtt{cons}_\alpha\mathtt{cons}_R(\mathtt{div}(\mathtt{tail}_{\beta_{n+1}}(\mathtt{cons}_{c_{n+1}}(N_{n+1}) - 2N_2), N_2)),\end{aligned}$$

and $\mathtt{cons}_\alpha\mathtt{cons}_R = \mathtt{cons}_\alpha([0,1])$, and

$$\begin{aligned}\mathtt{cons}_\alpha([0,1]) &= \left[\frac{\kappa_\alpha}{2}(0) + \frac{\kappa_\alpha}{2}, \frac{\kappa_\alpha}{2}(1) + \frac{\kappa_\alpha}{2}\right] \\ &= \left[\frac{\kappa_\alpha}{2}, \kappa_\alpha\right],\end{aligned}$$

hence $\kappa_{\left[\frac{\kappa_\alpha}{2}, \kappa_\alpha\right]} = \frac{\kappa_\alpha}{2} \leq \epsilon$.

A similar, and routine, analysis follows in the other cases. □

10.3 Root finding

In this section, we present a program for root finding. This program is based in the algorithm presented by Edalat [19]. It is a trisection method which can be considered as the redundant version of the well-known bisection method. Let $g : [-1,1] \to [-1,1]$ be a continuous function with a unique root in $(-1,1)$. We only consider the cases when g is increasing or decreasing, i.e.

Definition 10.3.1. *We say that g is:*

- increasing *if* $(\forall y \in [-1,1].y < x \Rightarrow g(y) < 0)$ *and* $(\forall y \in [-1,1].y > x \Rightarrow g(y) > 0)$,
- decreasing *if* $(\forall y \in [-1,1].y < x \Rightarrow g(y) > 0)$ *and* $(\forall y \in [-1,1].y > x \Rightarrow g(y) < 0)$.

The trisection algorithm is described as follows:

- We first try to establish whether $g(0)$ is greater or smaller than 0. If $g(0) > 0$ or $g(0) < 0$ then we can output $\mathtt{cons}_L$ or $\mathtt{cons}_R$ respectively.
- Otherwise, we know the sign of $c = g\left(-\frac{1}{2}\right)$ and $d = g(\frac{1}{2})$ and so
  - if c and d are greater than 0 then we output $\mathtt{cons}_R$ when g is decreasing and $\mathtt{cons}_L$ when g is increasing.
  - if c and d are smaller than 0 then we output $\mathtt{cons}_L$ when g is decreasing and $\mathtt{cons}_R$ when g is increasing.

– if c and d do not have the same sign, then we output $\texttt{cons}_C$.

Obviously after we output a partial result, we proceed the recursive definition with a careful but routine analysis of the parameters.

We now present the trisection program to compute recursively the root of a continuous function as specified above.

Notation 10.3.2. *In order to present a simplified version of our program, we use:*

```
cases                                   if a
   a : 1            instead of           then 1
   b : 2                                  else
   default : 3                             if b
                                              then 2
                                              else 3
```

$$\texttt{root} : ([-1,1] \to [-1,1]) \to [-1,1]$$

$$\texttt{root}(g) = \texttt{root}'\left(g \quad \left(-\frac{1}{2}\right) \quad 0 \quad \frac{1}{2}\right)$$

$$\texttt{root}' : ([-1,1] \to [-1,1]) \to [-1,1] \to [-1,1] \to [-1,1] \to [-1,1]$$

$\texttt{root}'(g\ x\ y\ z) =$
  cases
    $\texttt{check\_pos}(g,y) = -1 \ : \ \texttt{cons}_R\left(\texttt{root}'\left(g\ y\ \frac{(y+z)}{2}\ z\right)\right)$
    $\texttt{check\_pos}(g,y) = 0 \ :$
      cases
        $\texttt{check\_pos}(g,x) = -1 \ :$
          cases
            $\texttt{check\_pos}(g,z) = -1 \ : \ \texttt{cons}_R\left(\texttt{root}'\left(g\ z\ \frac{z+1}{4}\ \frac{z+1}{2}\right)\right)$
            $\texttt{check\_pos}(g,z) = 0 \ : \ \texttt{cons}_C\left(\texttt{root}'\left(g\ x\ \frac{y}{2}\ \frac{z}{2}\right)\right)$
            $\texttt{check\_pos}(g,z) = 1 \ : \ \texttt{cons}_C\left(\texttt{root}'\left(g\ \frac{x}{2}\ y\ \frac{z}{2}\right)\right)$
        $\texttt{check\_pos}(g,x) = 0 \ : \ \texttt{cons}_C\left(\texttt{root}'\left(g\ x\ \frac{y}{2}\ \frac{z}{2}\right)\right)$
        $\texttt{check\_pos}(g,x) = 1 \ :$
          cases
            $\texttt{check\_pos}(g,z) = -1 \ : \ \texttt{cons}_C\left(\texttt{root}'\left(g\ \frac{x}{2}\ y\ \frac{z}{2}\right)\right)$
            $\texttt{check\_pos}(g,z) = 0 \ : \ \texttt{cons}_C\left(\texttt{root}'\left(g\ x\ \frac{y}{2}\ \frac{z}{2}\right)\right)$
            $\texttt{check\_pos}(g,z) = 1 \ : \ \texttt{cons}_L\left(\texttt{root}'\left(g\ \frac{x-1}{2}\ \frac{x-1}{4}\ x\right)\right)$
    $\texttt{check\_pos}(g,y) = 1 \ : \ \texttt{cons}_L\left(\texttt{root}'\left(g\ x\ \frac{(x+y)}{2}\ y\right)\right)$

where `check_pos` is defined as:

$$\texttt{check\_pos} : ([-1,1] \to [-1,1]) \to [-1,1] \to \mathbb{N}$$

$$\begin{array}{l}
\texttt{check_pos}(g\ \ w)\ \ = \\
\quad \texttt{if}\ \ \texttt{rtest}_{a,b}(g(w)) \\
\quad\ \ \texttt{then} \\
\quad\quad \texttt{if}\ \ \texttt{rtest}_{a,0}(g(w)) \\
\quad\quad\quad \texttt{then}\ \ -1 \\
\quad\quad\quad \texttt{else}\ \ 0 \\
\quad\ \ \texttt{else} \\
\quad\quad \texttt{if}\ \ \texttt{rtest}_{0,b}(g(w)) \\
\quad\quad\quad \texttt{then}\ \ 0 \\
\quad\quad\quad \texttt{else}\ \ 1
\end{array}$$

with $-1 < a < 0 < b < 1$.

Following the same arguments as in the previous sections, it is easy to prove that the term `root` is convergent.

Part IV

Conclusions

Chapter 11

Summary of Work Done

In this chapter, we present the summary of the results presented in this work, which are presented in the second and third part of the thesis.

11.1 A Multi-valued Construction

Given a multi-valued construction, we have shown that it is possible to reconcile the aims of data abstraction, sequentiality and expressivity (at least for some computable first order functions, see Section 12.1) in a programming language for exact real-number computation.

We have presented a denotational semantics for the multi-valued or non-deterministic construction proposed by Boehm and Cartwright [10]. A powerdomain construction has been used to model the non-deterministic construction. Following the tradition in domain theory, we have used powerdomains to model the non-deterministic nature of the construction. We have shown that, among the well known powerdomains, neither the Plotkin nor the Smyth powerdomain can be used for this purpose. In fact, we have shown that not even other powerdomains, such as the Sandwich and the Mixed powerdomains, can be used for our purpose. Moreover, none of the known powerdomains can be used, unless they arise as the composition of powerdomains with the Hoare powerdomain as the last component in the composition.

11.2 A Sequential Program for Addition

Recalling the negative result of Escardó et al. [27] that "In the interval domain model addition is inherently parallel", we have been able to overcome the problem by considering a powerdomain of the interval domain as a model.

We have presented a sequential program for addition which in the case of total inputs, shows all the possible paths converging to the desired solution, i.e. a solution which is single-valued.

Furthermore, we have shown that it is possible to have sequential programs for important functions which admit only parallel realizations in the interval domain model. We have presented sequential programs for computable first order functions, such as multiplication, division, negation and absolute value. We have proved the correctness of these programs.

11.3 Recursive definitions

We have presented recursive definitions of first-order functions using the multi-valued construct and we have proved that, for total inputs, they produce the desired single-valued outputs. The definitions presented are: addition, multiplication, negation, absolute value and division.

11.4 The Programming Language

This is one of the main contributions of the thesis. We have presented a sequential programming language (called LRT) with a built-in abstract data type for real numbers. The language LRT can be seen from two different points of view:

- An extension of PCF [68] with the following additions: a ground type for real numbers and primitive operations for real-number computation, among which there is a non-deterministic constructor.
- A variation of Real PCF [24] with two primitive constructors removed from Real PCF: the parallel constructor pif and the test constructor head (also referred as $<_\perp$) and two primitive constructors added: the non-deterministic constructor rtest and the sequential conditional if.

Our main result for the language is computational adequacy between the operational and the denotational semantics of the language. To ensure adequacy, given the non-deterministic nature of the language, we introduce the notion of the operational meaning of a term, where, like the denotational values, the operational values are also taken in a powerdomain. The difference between the operational semantics and the denotational semantics is that the former is obtained by reduction but the latter is obtained by compositional means.

11.5 Total Correctness of Programs

By the results of Chapter 7, we are forced to use the Hoare powerdomain as a model for the programming language *LRT*. As is well known, Hoare models can be used only to prove the partial correctness of programs. In order to obtain the total correctness of programs from partial correctness, we introduce a generalization of the notion of termination in the case of real-number computation. This notion is needed because computations with real numbers are infinite, while computations with natural numbers are not. The recursive definitions discussed in Section 11.3 immediately give rise to *LRT* programs. Hence, we have proved with operational arguments that any of the programs defined above converge, and with denotational methods, that they converge to the desired value.

Chapter 12

Open Problems and Further Work

12.1 Expressivity of the language at first order types

Although we have shown a number of computable first order single-valued functions to be definable in *LRT*, we have not shown that all computable first order single-valued functions are definable in the language. By a straightforward extension of the results of [24], it is easily shown that every computable real number can be defined in the language. In addition, the definitions presented in Chapter 8 for the unit interval domain, can be easily extended to the whole interval domain. The previous observations and the fact that the four basic arithmetic operations are definable in the language allow us to conclude that at least all computable polynomial functions are definable in the language. The definability of all computable first order single-valued functions in *LRT* is still an open question, but we conjecture that these functions are definable. In this direction, we have at present some results.

In order to study definability in *LRT*, we consider a well known framework to exact real-number computation called computable analysis [86], where definability for the computable real numbers and the computable functions has been established. Furthermore, in this framework, it has been proved that signed digit representation (see Section 4.5.2) is an admissible naming system to define computable reals and computable functions.

Based on the theory of computable analysis, Weihrauch [86] has shown that in order to define the computable first order function $((\mathbb{R})^n \to \mathbb{R})$, it suffices to define all computable first order functions in signed digit representation $((3^\omega)^n \to 3^\omega)$. If we confine our attention to the computable function

of one argument, as a first step, we want to prove that for every computable function $\tilde{\phi} : 3^\omega \to 3^\omega$, there exists a corresponding definable single-valued function $\overline{f} : \mathcal{P}^H\mathcal{I}[-1,1] \to \mathcal{P}^H\mathcal{I}[-1,1]$ such that the following diagram commutes:

$$\begin{array}{ccccccc}
3^\omega & \xrightarrow{q} & [-1,1] & \xhookrightarrow{s} & \mathcal{I}[-1,1] & \xhookrightarrow{\eta} & \mathcal{P}^H\mathcal{I}[-1,1] \\
\downarrow{\scriptstyle \tilde{\phi}} & & \downarrow{\scriptstyle f} & & & & \downarrow{\scriptstyle \overline{f}} \\
3^\omega & \xrightarrow[q]{} & [-1,1] & \xhookrightarrow[s]{} & \mathcal{I}[-1,1] & \xhookrightarrow[\eta]{} & \mathcal{P}^H\mathcal{I}[-1,1]
\end{array}$$

Because there is no explicit type in LRT for streams, in order to represent elements of 3^ω we define programs of the type $\mathtt{nat} \to \mathtt{nat}$ in the language (the correct representation of elements of 3^ω is given by programs of type $\mathtt{nat} \to \mathtt{3}$, hence we assume that only three elements from the codomain are used). The interpretation in the model of such programs is given by functions of type $\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp$. Hence our main goal is: given a continuous single-value function $\phi : (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) \to (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp)$ for which there is a (unique) continuous function $\tilde{\phi} : 3^\omega \to 3^\omega$ such that Diagram 12.1 commutes and there is a (unique) continuous function $f : [-1,1] \to [-1,1]$ such that Diagram 12.2 commutes to define a functional
$F : ((\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) \to (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp)) \to (\mathcal{P}^H\mathcal{I}[-1,1] \to \mathcal{P}^H\mathcal{I}[-1,1])$
such that the following diagram commutes:

$$\begin{array}{ccccccccc}
(\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) & \xleftarrow{e} & 3^\omega & \xrightarrow[q]{} & [-1,1] & \xhookrightarrow[s]{} & \mathcal{I}[-1,1] & \xhookrightarrow{\eta} & \mathcal{P}^H\mathcal{I}[-1,1] \\
\downarrow{\scriptstyle \phi} & & \downarrow{\scriptstyle \tilde{\phi}} & & \downarrow{\scriptstyle f} & & & & \downarrow{\scriptstyle F(\phi)} \\
(\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) & \xleftarrow{e} & 3^\omega & \xrightarrow[q]{} & [-1,1] & \xhookrightarrow[s]{} & \mathcal{I}[-1,1] & \xhookrightarrow{\eta} & \mathcal{P}^H\mathcal{I}[-1,1]
\end{array}$$

That is to say, given any computable function from 3^ω to 3^ω we want to obtain an equivalent computable function from $\mathcal{P}^H[-1,1]$ to $\mathcal{P}^H[-1,1]$. Our proposed definition of the function F is given by:

$F\ (\phi)\ X\ =$
  let $d = \mathsf{head}(\phi(0^\omega))$ in
    if $\mathsf{forall}(\lambda\alpha.\mathsf{head}(\phi(\alpha)) == d)$
      then $\widehat{\mathsf{cons}}_d(F(\lambda\alpha.\mathsf{tail}(\phi(\alpha)))(X))$
      else

$$
\begin{array}{l}
\mathsf{if}\ \overline{\mathsf{rtest}}_{-1/2,1/2}(X) \\
\quad \mathsf{then} \\
\qquad \mathsf{if}\ \overline{\mathsf{rtest}}_{-1/2,0}(X) \\
\qquad\quad \mathsf{then}\ F(\lambda\alpha.\phi(\mathsf{cons}_L(\alpha)))(\widehat{\mathsf{tail}}_L(X)) \\
\qquad\quad \mathsf{else}\ F(\lambda\alpha.\phi(\mathsf{cons}_C(\alpha)))(\widehat{\mathsf{tail}}_C(X)) \\
\quad \mathsf{else} \\
\qquad \mathsf{if}\ \overline{\mathsf{rtest}}_{0,1/2}(X) \\
\qquad\quad \mathsf{then}\ F(\lambda\alpha.\phi(\mathsf{cons}_C(\alpha)))(\widehat{\mathsf{tail}}_C(X)) \\
\qquad\quad \mathsf{else}\ F(\lambda\alpha.\phi(\mathsf{cons}_R(\alpha)))(\widehat{\mathsf{tail}}_R(X))
\end{array}
$$

where forall [77] is defined by:

$$
\begin{array}{l}
\mathsf{witness\text{-}not} : ((\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{T}_\perp) \to \mathcal{P}^H\mathbb{T}_\perp) \to (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{T}_\perp) \\
\mathsf{witness\text{-}not}\ (P) = \\
\qquad \mathsf{let}\ w = \mathsf{witness\text{-}not}(\lambda\alpha.P(\widehat{\mathsf{cons}}_L(\alpha))) \\
\qquad\quad \mathsf{in\ if}\ P(\widehat{\mathsf{cons}}_L(w))\ \mathsf{then}\ \widehat{\mathsf{cons}}_R(\mathsf{witness\text{-}not}(\lambda\alpha.P(\widehat{\mathsf{cons}}_R(\alpha)))) \\
\qquad\qquad \mathsf{else}\ \widehat{\mathsf{cons}}_L(w)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{forall} : \ ((\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{T}_\perp) \to \mathcal{P}^H\mathbb{T}_\perp) \to \mathcal{P}^H\mathbb{T}_\perp \\
\mathsf{forall}\ (P) = \ P\ (\mathsf{witness\text{-}not}\ P)
\end{array}
$$

In other words, forall takes a total function $\phi : (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{T}_\perp) \to \mathcal{P}^H\mathbb{T}_\perp$, and if there exists $\alpha \in \mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{T}_\perp$ such that $\phi(\alpha) = \eta(\mathsf{false})$ then $[\![\mathsf{witness\text{-}not}]\!](\phi)$ is one such α; otherwise $[\![\mathsf{witness\text{-}not}]\!](\phi) = \mathsf{Y}\widehat{\mathsf{cons}}_R$.

The explanation of F is as follows: F compares the first digit obtained by applying the function ϕ to the input 0^ω with every possible first digit generated by applying ϕ to any other input. If the first digit is known to be the same for every possible output of ϕ, then the first digit of the result is output, otherwise, the input X is examined.

If X is examined, there are four cases which can apply at this stage. If X is known to be within the interval $[-1, 0]$ then we know that the function ϕ should look at all the inputs which have -1 as first element (equivalently cons_L). Hence, we call recursively the function F with first argument $\phi \circ \mathsf{cons}_L$ and with second argument $\widehat{\mathsf{tail}}_L(X)$ which is the remainder of X after looking at the first digit. The criteria for reasoning applied to the other cases follows similarly.

We have already the proof of the total correctness of the function forall. For the function F we want to prove the following:

Conjecture 12.1.1. *For any continuous single-valued function $\phi : (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) \to (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp)$ such that there exists a continuous function*

$\tilde{\phi} : 3^\omega \to 3^\omega$ *making Diagram 12.1 commute and such that there exists a continuous* $f : [-1,1] \to [-1,1]$ *making Diagram 12.2 commute, it holds that*

$$F(\phi)(\eta\{x\}) = \eta\{f(x)\}, \text{ for all } x \in [-1,1].$$

We have already proved that $F(\phi)(\eta\{x\}) \sqsubseteq \eta\{f(x)\}$. However the proof of $\eta\{f(x)\} \sqsubseteq F(\phi)(\eta\{x\})$ is still in progress.

A possible proof for $\eta\{f(x)\} \sqsubseteq F(\phi)(\eta\{x\})$ is as follows. Because $\eta\{f(x)\} = \bigsqcup\{\eta(b) \mid b \ll \{f(x)\}\}$, it is enough to show that for every $b \ll \{f(x)\}$ there is some n such that $\eta(b) \sqsubseteq F_n(\phi)(\eta\{x\})$. So we assume that $b \ll \{f(x)\}$. This amount to saying that $f(x) \in b^\circ$. Hence by continuity of f, there is an interval $c \in \mathcal{I}[-1,1]$ with $x \in c^\circ$ and $f(c^\circ) \subseteq b^\circ$. Because $\{x\}$ is a closed set as $[-1,1]$ is Hausdorff, the set $q^{-1}\{x\}$ is closed by continuity of q, and hence compact by the compactness of 3^ω. By the hypothesis that $f \circ q = q \circ \tilde{\phi}$, we have that $\tilde{\phi}(q^{-1}\{x\})$ is contained in the open set $q^{-1}(b^\circ)$. By continuity of $\tilde{\phi}$ and the fact that the sets $B_m(\alpha) = \{\beta \in 3^\omega \mid \forall i < m.\alpha_i = \beta_i\}$, $m \in \mathbb{N}$, form a base of the topology of 3^ω, we have that for each $\alpha \in q^{-1}\{x\}$ there is an $m_\alpha \in \mathbb{N}$ such that $\tilde{\phi}(B_{m_\alpha}(\alpha)) \subseteq q^{-1}(b^\circ)$. Now because $q^{-1}\{x\}$ is compact and $\{B_{k_\alpha}(\alpha) \mid \alpha \in q^{-1}\{x\}\}$ is an open cover of $q^{-1}\{x\}$, there is $F \subseteq q^{-1}\{x\}$ such that $\{B_{k_\alpha}(\alpha) \mid \alpha \in F\}$ is already a cover of $q^{-1}\{x\}$. For each $\alpha \in q^{-1}\{x\}$, we have $\tilde{\phi}(\alpha) \in q^{-1}(b^\circ)$, and hence there is a smallest number l_α such that $B_{l_\alpha}(\tilde{\phi}(\alpha)) \subseteq q^{-1}(b^\circ)$. Hence we have $\tilde{\phi}(B_{k_\alpha}(\alpha)) \subseteq B_{l_\alpha}(\tilde{\phi}(\alpha)) \subseteq q^{-1}(b^\circ)$. If it suffices to consider b of the form $q\{\beta' \in 3^\omega \mid \beta \leq \beta'\}$ for $\beta \in 3^*$, then we believe that a candidate for n would be $n = \mathsf{max}\{k_\alpha + \beta \mid \alpha \in F\}$ but the proof is still in progress.

The proof of convergence of the equivalent program for F and the generalization of ϕ to $\phi' : (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp)^n \to (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp)$ are left for further development.

$$\begin{array}{ccc} (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) & \xrightarrow{\phi} & (\mathcal{P}^H\mathbb{N}_\perp \to \mathcal{P}^H\mathbb{N}_\perp) \\ \uparrow e & & \uparrow e \\ 3^\omega & \xrightarrow{\tilde{\phi}} & 3^\omega \end{array} \tag{12.1}$$

$$\begin{array}{ccc} 3^\omega & \xrightarrow{\tilde{\phi}} & 3^\omega \\ \downarrow q & & \downarrow q \\ [-1,1] & \xrightarrow{f} & [-1,1] \end{array} \tag{12.2}$$

12.2 Denotational Semantics

We have seen that it is possible to have an expressive programming language for exact real-number computation with an abstract data type for real numbers and sequential operational semantics. However, it is still an open question whether it is possible to find a denotational semantics which would allow us to prove total correctness without the need to resort to operational methods, such as strong convergence.

As described above, in the interval domain model parallel operators are definable and, even if we consider a powerdomain of the interval domain as a model operational methods are needed. Although it is possible to prove total correctness in this way, it complicates the task of the programmers who want to verify the correctness of their programs, because they cannot rely completely on the mathematical model for the validity of their programs.

We draw attention to at least two possible directions to proceed:

- Looking for a new domain to represent real-numbers in domain theory. The interval domain has been the most widely used domain to represent real-numbers in domain theory. Being a continuous domain and representing all the elements of the real line, the interval domain is the ideal model to use. However in order to achieve sequentiality a new direction has to be taken.

- Using game semantics to present a model for real-numbers. Game semantics is an alternative technique for giving denotational semantics to programming languages; it would be interesting to know if there is a model in game semantics for the kind of language that we are looking at.

12.3 Full abstraction

Is LRT a fully abstract language?

Suppose we are given a semantics $[\![\cdot]\!]$ for LRT which interprets types by the Hoare powerdomain of posets. The interpretation $[\![\cdot]\!]$ is said to be fully abstract if the operational and denotational orderings coincide, that is, for every pair M, N of well-typed terms: $[\![M]\!] = [\![N]\!]$ if, and only if, $[M] = [N]$. We conjecture that the language LRT is not fully abstract.

12.4 Universality

Is *LRT* a universal language? We have said that a language is universal if every computable element of its domain of definition is definable on it. Because it is not clear what the notion of computability should be in the multi-valued setting, we do not have an answer to the above question. This also has to be investigated. There are some proposals in this direction by Weihrauch [86], but it is also not clear whether his considerations apply in our setting. Because we are dealing with multi-valued functions, we conjecture that the language *LRT* is represented by the dashed polygon in Figure 12.1 and is not universal.

In the figure, the language Real PCF+ is the language Real PCF with an existential quantifier added. It was showed by Escardó [24] that this language is universal, in the sense that every computable element of its universe of discourse is definable in the language. Real PCF is contained in Real PCF+, because there exists at least a computable function called $\exists$ which is not definable in Real PCF, and so is not universal (for the construction of this $\exists$ see Plotkin [68]). There is a weaker version called weak Real PCF, which is Real PCF without the parallel conditional `pif`. Farjudian [29] has shown that all first order weak Real PCF definable functions are Vuillemin sequential [3] (moreover, his arguments shows that any extension of the language with continuous first-order unary functions has the same property).

12.5 Efficiency

Is *LRT* an efficient language?

One of the major concerns in exact real-number computation is the lack of an efficient programming language or calculator to compute with real numbers (it is believed that the price of being inefficient is a consequence of guaranteeing correctness in the solution). For example, although Real PCF is universal, in practice it is inefficient (at each step of the computation of some computable first order function, at least three new parallel processes are needed to compute the answer and for second order computable functions the situation is even worse).

There have been other implementations [69, 58, 59], which have been proved to be more efficient for particular programs, but not in general. An implementation of exact-real number computation using linear fractional transformations is believed to be more efficient, although there is no formal proof of such a contention. Heckmann [38, 39], has studied this approach in more detail with some interesting results.

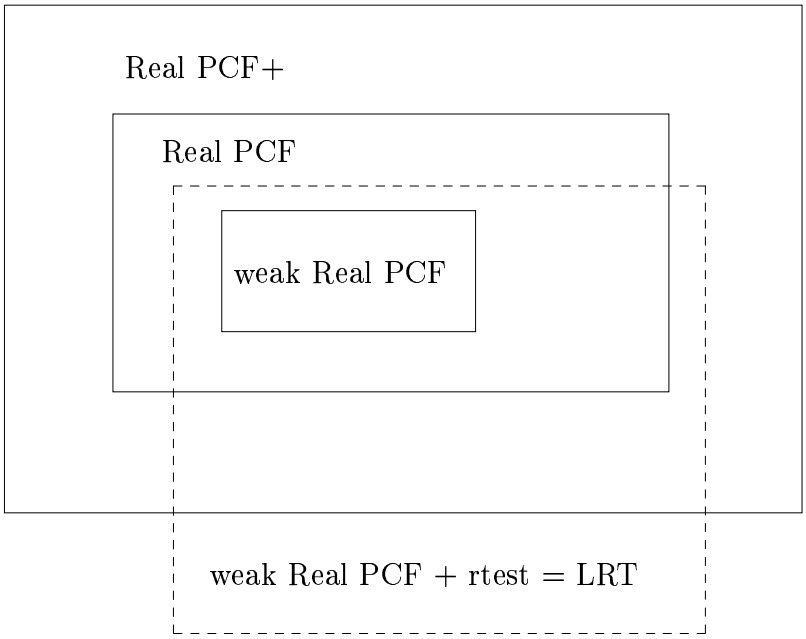


Figure 12.1: Expressiveness of Languages

It would be useful to implement the programming language *LRT* to compare its efficiency with previous approaches. Being a sequential language, *LRT* is, we believe, more efficient than Real PCF but we do not have any clue to its behaviour with respect to the other approaches.

12.6 A different language

A new syntax would be obtained if we added the power constructor $\mathcal{P}$ as a data type to the language.

The denotational semantics given to the language *LRT* in Chapter 9 is in terms of the Hoare powerdomain. If the syntax of the language is modified in such a way that the power constructor $\mathcal{P}\sigma$ is included as an element of it in the following way:

$$\sigma ::= \ldots \mid \mathcal{P}\sigma \mid \ldots,$$

it would be interesting to present a denotational semantics for such a language because we may be able to interpret programs for example of type $(\mathtt{I} \to \mathtt{I}) \to \mathtt{I}$ as $(\mathcal{I} \to \mathcal{I}) \to \mathcal{P}\mathcal{I}$ when required, instead of the actual interpretation $(\mathcal{P}\mathcal{I} \to \mathcal{P}\mathcal{I}) \to \mathcal{P}\mathcal{I}$.

12.7 A different paradigm

Levy [55] introduced a new programming language paradigm called call-by-push-value (CBPV). His work is based on Moggi and Filinski. The interesting part of CBPV for this thesis, it seems to me, is the distinction of types in the syntax of the language defined. Under the slogan "a value is, a computation does", Levy distinguishes what he called values from computations in the type system. The two classes of types presented in CBPV are:

$$\begin{array}{lrcl} \text{value types} & A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \\ \text{computation types} & B & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

The value types contain the usual: product type $A \times A$, the unary type 1 and the sum (possibly infinite) of types $\sum_{i\in I} A_i$. In the computation types the usual (possibly infinite) product $\prod_{i\in I} \underline{B}_i$ is considered.

The interesting types in our setting would be $U\underline{B}$, FA and $A \to \underline{B}$. The explanation given of these types, according to Levy, is as follows:

- A value of type $U\underline{B}$ is a thunk of a computation of type $\underline{B}$. In our setting, if we consider that a computation of type $\underline{B}$ lives in the Hoare powerdomain of a cpo, this type would allow us to define constructors in the language from computations to values.

- Given that the computational effect in the language is nondeterminism, a computation of type FA would be a value of $\mathcal{P}A$.

- A computation of type $A \to \underline{B}$ takes a value and then returns a computation of type $\underline{B}$.

This distinction of types would allow us to present a denotational semantics in the usual domain theoretical sense with FA interpreted as $\mathcal{P}^H A$. The interesting problem would be to present the meaning of a thunk of a computation in real number computation. It is compelling because we have only partial information about the result which we are working on, since computations with reals are infinite, whereas for example the flat domain of natural numbers is finite. This raises another open question: that of defining a call-by-value semantics for real number computation.

Bibliography

[1] S. Abramsky. Experiments, powerdomains and fully abstract models for applicative multiprogramming. In M. Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1983.

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

[3] R. Amadio and P.-L. Curien. *Domains and Lambda Calculi*, volume 46. Cambridge University Press, 1998.

[4] S.O. Anderson and A.J. Power. A representable approach to finite nondeterminism. *Theoretical Computer Science*, 177(1):3–25, 1997.

[5] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961.

[6] H. P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

[7] M. Beeler, R. W. Gosper, and R. Schroeppel. Continued fraction arithmetic. HAKMEN, MIT AI Memo 239, February 1972.

[8] E. Bishop and D. Bridges. *Constructive Analysis*. Springer-Verlag, Berlin, 1985.

[9] H. J. Boehm. Constructive real interpretation of numerical programs. *SIGPLAN Notices*, 22(7):214–221, 1987.

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

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

[12] L.E.J. Brouwer. Besitzt jede reelle zahl eine dezimalbruchentwicklung? *Math Ann*, 83:201–210, 1920.

[13] P. Buneman, S. Davidson, and A. Watters. A semantics for complex objects and approximate queries. In *Principles of Database Systems*, pages 305–314. ACM, 1988.

[14] J.-M. Chesneaux and J. Vignes. Les fondements de l'arithmétique stochastique. *C.R. Acad. Sci. Paris Sér. I Math.*, 315(13):1435–1440, 1992.

[15] P. Di Gianantonio. *A functional approach to computability on real numbers*. PhD thesis, University of Pisa, 1993. Thecnical Report TD 6/93.

[16] P. Di Gianantonio. A golden ratio notation for the real numbers. Technical Report CS-R9602, CWI, Amsterdam, 1996.

[17] P. Di Gianantonio. Real number computability and domain theory. *Information and Computation*, 127(1):12–25, May 1996.

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

[19] A Edalat. Two algorithms for root finding in exact real arithmetic. *Third Real Numbers and Computers Conference*, pages 27–44, 1998.

[20] A. Edalat and P. J. Potts. A new representation for exact real numbers. In S. Brookes and M. Mislove, editors, *Electronic Notes in Theoretical Computer Science*, volume 6. Elsevier, 2000.

[21] A. Edalat, P. J. Potts, and P. Sünderhauf. Lazy computation with exact real numbers. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, pages 185–194. ACM, 1998.

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

[23] 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.

[24] M.H. Escardó. *PCF Extended with Real Numbers: a domain-theoretic approach to higher-order exact real number computation.* PhD thesis, University of London, 1997.

[25] M.H. Escardó. Effective and sequential definition by cases on the reals via infinite signed-digit numerals. In *Third Workshop on Computation and Approximation (Comprox III)*, volume 13 of *Electronic Notes in Theoretical Computer Science*, 1998.

[26] M.H. Escardó. Introduction to exact numerical computation. Notes for a tutorial course at ISSAC 2000, August 2000.

[27] M.H. Escardó, 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).

[28] 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.

[29] A. Farjudian. Sequentiality and piecewise affinity in fragments of Real-PCF. To Appear in ENTCS, 2003.

[30] A. Farjudian. *Sequentiality in Real Number Computation.* PhD thesis, University of Birmingham, August 2004.

[31] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous latticces and domains.* Cambridge University Press, 2003.

[32] C. A. Gunter. Relating total and partial correctness interpretations of non-deterministic programs. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 306–319. ACM Press, 1990.

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

[34] C. A. Gunter and D. S. Scott. Semantic domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B of *Formal Models and Semantics*, pages 633–674. Elsevier Science Publishers, Amsterdam, 1990.

[35] R. Heckmann. *Power Domain Constructions.* PhD thesis, Universität des Saarlandes, 1990.

[36] R. Heckmann. Set domains. In N. Jones, editor, *ESOP'90*, volume 432 of *Lecture Notes in Computer Science*, pages 177–196. Springer-Verlag, 1990.

[37] R. Heckmann. How many argument digits are needed to produce n result digits? In Abbas Edalat, David Matula, and Philipp Sünderhauf, editors, *Electronic Notes in Theoretical Computer Science*, volume 24. Elsevier, 2000.

[38] Reinhold Heckmann. The appearance of big integers in exact real arithmetic based on linear fractional transformations. In *Proceedings of the First International Conference on Foundations of Software Science and Computation Structure*, volume 1378, pages 172–188. Springer, 1998.

[39] Reinhold Heckmann. Contractivity of linear fractional transformations. *Theoretical Computer Science*, 279, (1):65–82, 2002.

[40] M. C. B. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In J. Beçvar, editor, *Mathematical Foundations of Computer Science 1979*, volume 74, pages 108–120. Springer-Verlag, 3–7 September 1979.

[41] M.C.B. Hennessy. The semantics of call-by-value and call-by-name in a nondeterministic environment. *SIAM journal of Computing*, 9(1):67–84, Febrary 1980.

[42] M.C.B. Hennessy. Powerdomains and nondeterministic recursive definitions. In Dezani-Ciancaglini and Montanari, editors, *Proceedings, International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 1982.

[43] A. Jeffrey. A fully abstract semantics for a higher-order functional language with nondeterministic computation. *Theoretical Computer Science*, 228(1-2):105–150, 1999.

[44] J.L. Kelley. *General Topology.* D. van Nostrand, New York, 1955.

[45] D. E. Knuth. *The Art of Computer Programming, Seminumerical algorithms*, volume 2. Addison-Wesley, 2nd edition, 1981.

[46] Ker-I Ko. *Complexity Theory of Real Functions.* Birkhauser Boston, Boston, 1991.

[47] H. Kopka and P. W. Daly. *A Guide to LaTeX*. Addison-Wesley, third edition, 1999.

[48] P. Kornerup and D.W. Matula. An on-line arithmetic unit for bit-pipelined rational arithmetic. *Journal of Parallel and Distributed Computing*, 5(3):310–330, 1988.

[49] P. Kornerup and D.W. Matula. Exploiting redundancy in bit-pipelined rational arithmetic. In M.D. Ercegovac and E. Swartzlander, editors, *Proceeding of the 9th IEEE Symposium on Computer Arithmetic*, pages 199–126. IEEE Computer Science Press, 1989.

[50] P. Kornerup and D.W. Matula. An algorithm for redundant binary bit-pipelined rational arithmetic. *IEEE Transactions on Computers*, C-39(8):1106–1115, 1990.

[51] P. Kornerup and D.W. Matula. An algorithm for redundant binary bit-pipelined rational arithmetic. *IEEE Transactions on Computers*, 39(8):1106–1115, August 1990.

[52] P. Kornerup and D.W. Matula. Finite precision lexicographic continued fraction number systems. In *Proceedings of the* 7^{th} *IEEE Symposium on Computer Artihmetic*, pages 207–214, Urbana, 1995. IEEE Computer Society Press.

[53] C. Kreitz and K. Weihrauch. Theory of representations. *Theoretical Computer Science*, 38(1):35–53, 1985.

[54] D. R. Lester. Vuillemin's exact real arithmetic. In R. Heldal, C.K. Holst, and P.L. Wadler, editors, *Proceedings of the 1991 Glasgow Workshop on Functional Programming*, pages 225–238, Berlin, 1992. Springer-Verlag.

[55] P. B. Levi. *Call-By-Push-Value A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Kluwer Academic Publishers, 2004.

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

[57] J.R. Marcial-Romero and M.H. Escardó. Semantics of a sequential language for exact real-number computation. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 426–435. IEEE Computer Society, July 2004.

[58] D. McGaw. Exact arithmetic using golden ratio. 4th Year Project Report, Department of Computer Science, University of Edinburgh, 1999.

[59] V. Ménissier-Morain. Arbitrary precision real arithmetic: Design and algorithms. Submitted to the Journal of Symbolic Computation, September 1996.

[60] R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, 1966.

[61] N.Th. Müller. Subpolynomial complexity classes of real functions and real numbers. In Laurent Knott, editor, *International Colloquium on Automata, Languages, and Programming*, volume 226, pages 284–293. Spriger-Verlag New York, Inc., 1986.

[62] N.Th. Müller. The iRRAM: Exact arithmetic in C++. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis*, volume 2064 of *Lecture Notes in Computer Science*, pages 222–252, Berlin, 2001. Springer. 4th International Workshop, CCA 2000, Swansea, UK, September 2000.

[63] A. M. Nielsen and P. Kornerup. MSB-first digit serial arithmetic. *Journal of Universal Computer Science*, 1(7):527–547, July 1995.

[64] D. Normann. Exact real number computations relative to hereditarily total functionals. *Theoretical Computer Science*, pages 437–453, 2002.

[65] S.L. Peyton-Jones. Arbitrary precision arithmetic using continued fractions. INDRA Note 1530, University College London, 1984.

[66] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh, 1981.

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

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

[69] Dave Plume. A calculator for exact real number computation. 4th Year Project Report, Department of Computer Science and Artificial Intelligence, University of Edinburgh, 1998.

[70] P. Potts. *Exact Real Arithmetic using Möbius Transformations.* PhD thesis, Imperial College, 1998.

[71] P. Potts and A. Edalat. Exact real computer arithmetic. Departamental Technical Report DOC 97/9, Imperial College, March 1997.

[72] P. J. Potts, A. Edalat, and M.H Escardó. Semantics of exact real arithmetic. In *In Proceedings of the Twelveth Annual IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press, 1997.

[73] A. Schalk. *Algebras for Generalized Construction*. Doctoral thesis, TU Darmstadt, 1993. 174 pp.

[74] D. S. Scott. Lattice theory, data type and semantics. In *Formal semantics of programming languages*, pages 66–106. Englewood Cliffs, Prentice-Hall, 1972.

[75] P. Sünderhauf. Incremental addition in exact real arithmetic. In R. Harmer et al., editor, *Advances in Theory and Formal Methods of Computing: Proceedings of the fourth Imperial College Workshop*. IC Press, 1997.

[76] A. Simpson. The convex powerdomain in a category of posets realized by cpos. In *Category Theory and Computer Science, 6th International Conference, CTCS '95, Cambridge, UK, Proceedings*, volume 953 of *Lecture Notes in Computer Science*, pages 117–145. Springer, August 1995.

[77] A Simpson. Lazy functional algorithms for exact real functionals. *Mathematical Foundations of Computer Science*, 1450:456–464, 1998.

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

[79] M. B. Smyth. Power domains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Theoretical Computer Science*, pages 662–675. Springer-Verlag, 1983.

[80] W.A. Sutherland. *Introduction to Metric and Topological Spaces*. Oxford Science Publications, 1975.

[81] R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.

[82] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Math. and Comp. in Simul*, 35(3):233–261, 1993.

[83] J. Vuillemin. Exact real computer arithmetic with continued fractions. In *Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 14–27. ACM Press, 1988.

[84] O. Watanuki and M.D. Ercegovac. Error analysis of certain floating-point on-line algorithms. *IEEE Transactions on Computers*, 32(4):352–358, April 1983.

[85] K. Weihrauch. *Computability.* Springer-Verlag, Berlin, 1987.

[86] K. Weihrauch. *Computable Analysis.* Springer, 2000.

[87] K. Weihrauch and C. Kreitz. Representations of the real numbers and the open subsets of the set of real numbers. *Annals of Pure and Applied Logic*, 35:247–260, 1987.

[88] E. Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.

Index