PCF extended with real numbers

Martín Hötzel Escardó

Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, United Kingdom, E-mail: `mhe@doc.ic.ac.uk`

Abstract

We extend the programming language PCF with a type for (total and partial) real numbers. By a *partial real number* we mean an element of a cpo of intervals, whose subspace of maximal elements (single-point intervals) is homeomorphic to the Euclidean real line. We show that partial real numbers can be considered as "continuous words". Concatenation of continuous words corresponds to refinement of partial information. The usual basic operations cons, head and tail used to explicitly or recursively define functions on words generalize to partial real numbers. We use this fact to give an operational semantics to the above referred extension of PCF. We prove that the operational semantics is sound and complete with respect to the denotational semantics. A program of real number type evaluates to a head-normal form iff its value is different from $\perp$; if its value is different from $\perp$ then it successively evaluates to head-normal forms giving better and better partial results converging to its value.

Keywords: Exact real number computation. Domain Theory and Interval Analysis. Denotational and operational semantics.

1 Introduction

There are several practical and theoretical approaches to exact real number computation (see eg. [7–9,18,21,24,26,30,31,37,39,40,42]). However, the author is not aware of any attempt to give denotational and operational semantics to an implementable programming language with a data type for exact real numbers. Most approaches to exact real number computation are based on representation of real numbers in other data types, such as streams of digits or rational numbers. P. di Gianantonio [9] discusses an extension of PCF with streams and shows how to represent real numbers in this extension (see below).

In this paper we extend the programming language PCF [28] with a type for real numbers. This type is interpreted as the cpo of intervals introduced independently by R.M. Moore in the 60's [25] and by D.S. Scott in the

To appear in *Theoretical Computer Science*, July 1996

early 70's [32]. In fact, such an extension was one of the problems left open by G.D Plotkin [28]. It is straight-forward to give a denotational semantics to such an extension, but it is not immediate how to give an operational semantics to it. P. di Gianantonio also presents an extension of PCF with a ground type interpreted as an algebraic cpo of real numbers, but he does not give an operational semantics to it.

An important feature of our approach to exact real number computation is that 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. Of course, the PCF interpreter has access only to concrete representations. The correct interaction between the abstract level and the concrete level is usually referred to as the *Adequacy Property* (of the operational semantics with respect to the denotational semantics). At the denotational level, a PCF program is just a mathematical expression denoting a number or a function. The task of the programmer is to find a mathematical expression denoting the entity that he or she has in mind. This entity is usually given by unrestricted mathematical means. In this case the programmer has to find an equivalent PCF expression. The Adequacy Property ensures that the entity denoted by the program will actually be computed by the PCF interpreter, and this is why the programmer is not concerned with representations.

We refer to the elements of the cpo of intervals as "partial real numbers". The domain of partial numbers is a non-algebraic continuous cpo. Its subspace of maximal elements (single-point intervals) is homeomorphic to the Euclidean real line, so that real numbers are special cases of partial real numbers. Notice that no algebraic cpo can have this property.

We show that partial real numbers can be considered as "continuous words", in the sense that they can be given the structure of a monoid, in such a way that it has a submonoid isomorphic to the monoid of words over any finite alphabet. Moreover, as it is the case for words, the prefix preorder of the monoid of continuous words coincides with its information order. This coincidence is the basis for the successful interaction between the operational and denotational semantics of PCF extended with real numbers.

Concatenation of continuous words have intuitive geometrical and computational interpretations. Geometrically, a concatenation of continuous words corresponds to a rescaling of an interval followed by a translation (an affine transformation). Computationally, a concatenation of continuous words corresponds to refinement of partial information; in a concatenation xy the partial real number y refines the information given by x, by selecting a subinterval of x.

The notion of length of words generalizes to partial real numbers. The length of a partial real number is an extended non-negative real number, being infinity iff the partial number is maximal, and zero iff the partial number is bottom. Roughly speaking, the length of a partial number x considered as a partial realization of an unknown real number y gives the number of digits of an expansion of y that x is able to give correctly. The concatenation operation "adds lengths", in the sense that the length function is a monoid homomorphism from the monoid of partial real numbers to the additive monoid of non-negative extended real numbers.

The usual basic operations cons, head and tail used to explicitly or recursively define functions on words generalize to partial real numbers. Geometrically, the operation cons is an affine transformation (in analytic terms, a linear map) on the unit interval, tail reverses the effect of cons, and head decides in which half of the real line its argument lies, telling us nothing in ambiguous cases.

Concatenation of partial numbers can be infinitely iterated. This fact gives a notion of meaningful infinite computation. The concatenation of finite initial segments of infinite computations gives more and more information about the final result, in such a way that every piece of information about the (ideal) result is eventually produced in a finite amount of steps. In practice, the terms of a computation can be taken as the partial numbers with distinct rational end-points.

The interpretation of partial real numbers as continuous words is related to a well-known approach to real number computation. Usual binary expansions are not appropriate representations for real number computation; for instance, multiplication by three is not computable if we read the expansions from left to right [42]. But binary expansions of numbers in the signed unit interval $[-1, 1]$ allowing a digit -1 turn out to be effective [8,9,17,40–42]. In the domain of partial numbers contained in the signed unit interval, infinite concatenations of the partial numbers $[-1, 0]$, $[-\frac{1}{2}, \frac{1}{2}]$ and $[0, 1]$ correspond to binary expansions of numbers in the signed unit interval using the digits -1, 0 and 1 respectively.

We use the fact that partial real numbers can be considered as continuous words to obtain an operational semantics for the extension of PCF referred above. We prove that the operational semantics enjoys the following Adequacy Property: a program of real number type evaluates to a head-normal form iff its value is different from $\perp$; if its value is different from $\perp$ then it successively evaluates to head-normal forms giving better and better partial results converging to its value.

To be accurate, the interpretation of partial numbers as continuous words holds only for the domain of partial real numbers contained in the unit interval (or any other compact interval). The domain of partial number contained in the unit interval is referred to as the *lazy unit interval*, and the domain of all partial real numbers is referred to as the *lazy real line*. The above results are extended from the lazy unit interval to the lazy real line via an action of the lazy unit interval on the lazy real line.

Organization

In Section 2 we briefly introduce the domain-theoretic and topological aspects of partial real numbers which are relevant to this paper. In Section 3 we develop the idea that "partial real numbers are continuous words". In Section 4 we extend PCF with partial real numbers. Finally, in Section 5 we present some concluding remarks and directions for further work.

2 The domain of partial real numbers

2.1 Domains

Our main references to Domain Theory are [3] and [16]. The widely circulated notes [29] are an excellent introduction, but they do not cover (non-algebraic) continuous cpos. The book [19] contains all the domain-theoretic material needed to understand PCF (without real numbers) as well as a detailed account to PCF. In this paper a *(continuous Scott) domain* is defined to be a bounded complete ω-continuous cpo; notice that a *Scott domain* is usually defined to be a bounded complete ω-*algebraic* cpo. In Section 3 we make use of elementary domain theory only, and therefore the reader can think of a domain just as a bounded complete cpo. But in Section 4 we make essential use of continuity, implicitly in the use of function spaces and explicitly in the proof of the Adequacy Theorem 34.

When we refer to a dcpo as a topological space we mean its set of elements under its Scott topology [3,16,35].

2.2 The lazy real line

We denote by $\mathcal{R}$ the set of non-empty compact subintervals of the real line ordered by $x \sqsubseteq y$ iff $x \supseteq y$; see Moore [25] and Scott [32]. If we add a bottom element to the continuous dcpo $\mathcal{R}$, which can be concretely taken as the interval $(-\infty, +\infty)$, then $\mathcal{R}$ becomes a domain, referred to as the *lazy real line.*

For the topological connections between Domain Theory and Interval Analysis [25] see [15]. In particular, it is shown that the Scott open sets are the Moore open upper sets. Since Moore restricts himself to monotone functions, the results presented in [25] go through if we replace the Moore topology by the Scott topology. See also [4] for more connections between Domain Theory and Interval Analysis.

Following the points of view of both Moore and Scott, we do not consider the elements of $\mathcal{R}$ as intervals. We instead consider them as generalized real numbers, in a similar way that complex numbers are considered as generalized real numbers, and we call them *partial real numbers*. The maximal partial real numbers (that is, the singleton intervals) are identified with the real numbers. This identification makes sense from a topological view, because the subspace of maximal elements of $\mathcal{R}$ is homeomorphic to the Euclidean real line. Non-maximal partial real numbers can be considered as partial realizations of real numbers. For example, $[3.14, 3.15]$ can be considered as a partial realization of π (and of several other numbers). Hence the designation *partial real number*.

A subset of the lazy real line has a join iff it has an upper bound; when the join exists, it is the intersection of the subset. Every subset of the lazy real line has a meet, namely the least interval (most defined partial number) containing the union of the set.

We denote the left and right end-points of a partial number x by $\underline{x}$ and $\overline{x}$ respectively, so that $x = [\underline{x}, \overline{x}]$.

Two partial numbers x and y have an upper bound iff they overlap. In this case their join is the overlapping part; that is, $x \sqcup y = [\max(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})]$. Dually, we have that their meet is given by $x \sqcap y = [\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})]$.

The way-below relation on the lazy real line is given by $x \ll y$ iff $x = \bot$, or $\underline{x} < \underline{y}$ and $\overline{y} < \overline{x}$. Therefore, a basis for the lazy real line is given by the set containing $\bot$ and the non-maximal partial numbers with rational (respectively dyadic) end-points. Recall that a dyadic number is a rational of the form $m/2^n$.

2.3 Order of magnitude on the lazy real line

We define a strict order $<$ on partial numbers (not to be confused with the strict version $\sqsubset$ of the information order or with the way-below order $\ll$) by $x < y$ iff $\overline{x} < \underline{y}$. This relation is clearly irreflexive, transitive, and asymmetric, in the sense that $x < y$ together with $x > y$ is impossible.

We say that two elements x and y of a domain are *consistent*, written $x \simeq y$, if they have a common upper bound. The consistency relation is always reflexive and symmetric, but not transitive, and it is preserved by monotone maps.

The following proposition is immediate:

Proposition 1 *For all partial real numbers x and y, exactly one of the relations $x < y$, $x \simeq y$ and $x > y$ holds.*

We define a relation $\leq$ on partial numbers (not to be confused with the information order $\sqsubseteq$) by $x \leq y$ iff $\overline{x} \leq \underline{y}$. This relation does not enjoy any of the properties that an order relation should satisfy, and therefore the notation can be misleading. We introduce it only for notational convenience.

2.4 The lazy unit interval

The set $\mathcal{I}$ of all partial numbers contained in the unit interval $[0, 1]$ is a domain, referred to as the *lazy unit interval.* The bottom element of $\mathcal{I}$ is the partial number $[0, 1]$. Its way-below order is given by $x \ll y$ iff $\underline{x} = 0$ or $\underline{x} < \underline{y}$, and $\overline{y} < \overline{x}$ or $\overline{y} = 1$.

The lazy unit interval can be presented in a geometrically more convenient form as follows. The unit square $[0, 1] \times [0, 1]$ under the componentwise order induced by the usual order $\leq$ on $[0, 1]$ is a continuous lattice, whose Lawson topology coincides with the Euclidean topology on the unit square; see [16] for a proof of these facts. If we consider the points below (equivalently, on the left of) the diagonal which goes from $(0, 1)$ to $(1, 0)$, that is, the points (x, y) with $x + y \leq 1$, we get a triangle, which we refer to as the *unit triangle.* The unit triangle is easily seen to be a domain. Its maximal elements are the points (x, y) with $x + y = 1$, that is, the points on the diagonal.

It turns out that the unit triangle is isomorphic to the lazy unit interval. The isomorphisms can be taken as $(x, y) \mapsto [1 - y, x]$ and $[x, y] \mapsto (y, 1 - x)$. We can think of the unit triangle as a *coordinate system* for the lazy unit interval.

We have a similar fact for the lazy real line; a coordinate system for $\mathcal{R}$ is given by the half-plane consisting of the points (x, y) with $x + y \leq 0$. A coordinate system for $\mathcal{R}_\perp$ is obtained by adding a point $(-\infty, -\infty)$ to the half-plane.

3 Partial real numbers considered as continuous words

Subsections 3.1–3.7 are restricted to the lazy unit interval and Subsection 3.8 extends the results of these sections to the lazy real line.

3.1 The prefix preorder of a monoid

Recall that a monoid is a set together with a binary associative operation and a neutral element for this operation. It is customary to refer to this operation as multiplication, but in this paper it is convenient to refer to it as *concatenation*.

By a word over an alphabet Σ we mean either a finite word in Σ^* or an infinite word in Σ^ω. We denote the set of all words by Σ^∞. Usual concatenation of words, with the convention that $xy = x$ if x is infinite, makes Σ^∞ into a monoid with neutral element the empty word ε.

In any monoid $(M, \cdot, e)$ we can define a preorder, called its *prefix preorder*, by $x \leq z$ iff $xy = z$ for some y. In this case x is called a *prefix* of z and y is called a *suffix*. This relation is reflexive because e is right neutral, and it is transitive because the concatenation operation is associative. It has e as its least element because e is left neutral. Monoid homomorphisms preserve the least element and the prefix preorder, by the very definition of monoid homomorphism. An element x is maximal iff it is *left dominant*, in the sense that $xy = x$ for every y. The meet of a set, when it exists, is the greatest common prefix of the elements of the set.

The prefix preorder of the monoid Σ^∞ makes the set Σ^∞ into a Scott domain [35]. In particular, the prefix preorder is a partial order; that is, it is antisymmetric. An element of Σ^∞ is maximal iff it is an infinite word, and it is finite in the domain-theoretic sense iff it is a finite word. The concatenation operation seen as a function $\Sigma^\infty \times \Sigma^\infty \to \Sigma^\infty$ is not continuous, because it is not even monotone. But it is continuous on its second argument. This can be expressed by saying that left translations $x \mapsto ax$ are continuous for all words a.

3.2 Left translations of a monoid

We denote a left translation $x \mapsto ax$ of a monoid $(M, \cdot, e)$ by cons_a. Left translations are monotone and $\mathrm{cons}_a(M) = \uparrow a$, where $\uparrow a = \{x \in M | a \leq x\}$. An element a is left cancelable iff the translation cons_a is injective.

If x is left cancelable and $x \leq z$, we denote the unique y such that $xy = z$ by z/x, so that $x(z/x) = z$. The basic properties of this (partially defined) quotient operation are:

$$x/e = x$$
$$x/x = e$$

$$(y/x)(z/y) = z/x$$
$$(xy)/z = (x/z)y$$

when the quotients are defined.

Let a be a left cancelable element of a monoid M. Then the co-restriction of cons_a to its image is a bijection between the sets M and $\uparrow a$, with inverse $x \mapsto x/a$. Therefore M is a preordered set isomorphic to the set $\uparrow a$ under the inherited order, because cons_a is monotone.

The left cancelable elements of Σ^∞ are the finite words. It follows that $\uparrow a$ is a domain isomorphic to Σ^∞ for every finite word a. The subsection below shows that the lazy unit interval has the same property for every non-maximal partial number a, for a suitable concatenation operation on partial numbers.

3.3 Concatenation of partial real numbers

Define a binary operation $(x, y) \mapsto xy$ on $\mathcal{I}$ by

$$xy = [(\overline{x} - \underline{x})\underline{y} + \underline{x},\ (\overline{x} - \underline{x})\overline{y} + \underline{x}]$$

That is, given $x, y \in \mathcal{I}$, rescale and translate the unit interval so that it becomes x, and define xy to be the interval which results from applying the same rescaling and translation to y. Then it is immediate that xy is a subinterval of x. The rescaling factor is the diameter of x, namely $\overline{x} - \underline{x}$, and the translation constant is the left end-point of x. If x is maximal, then its diameter is zero, so that $xy = x$. In order to simplify notation, we let κ_x stand for the diameter of x and μ_x stand for the left end-point of x. This is the notation of [23] for affine transformations. Notice that $x = [\mu_x, \mu_x + \kappa_x]$. Then we can write

$$xy = \kappa_x y + \mu_x$$

Theorem 2 $(\mathcal{I}, \cdot, \bot)$ *is a monoid with the following properties:*

(i) Its prefix preorder coincides with the information order of the domain $\mathcal{I}$.
(ii) Its left cancelable elements are the non-maximal ones.
(iii) Its left translations preserve all meets and all existing joins.

Proof. Let $x, y, z \in \mathcal{I}$. Then

$$xy = \kappa_x y + \mu_x = \kappa_x[\mu_y,\ \mu_y + \kappa_y] + \mu_x = [\kappa_x\mu_y + \mu_x,\ \kappa_x(\mu_y + \kappa_y) + \mu_x]$$

Hence $\mu_{xy} = \kappa_x \mu_y + \mu_x$ and $\kappa_{xy} = \kappa_x \kappa_y$. Therefore

$$(xy)z = \kappa_{xy} z + \mu_{xy} = \kappa_x \kappa_y z + \kappa_x \mu_y + \mu_x = \kappa_x(\kappa_y z + \mu_y) + \mu_x = x(yz)$$

Now, clearly $x\bot = x$, because by definition $\bot = [0, 1]$ is rescaled and translated so that it becomes x. Also, $\bot x = x$ because by definition $\bot = [0, 1]$ is rescaled and translated so that it becomes itself; hence $\bot x$ is the application of the identity to x. Therefore $(\mathcal{I}, \cdot, \bot)$ is a monoid.

(i) By construction, $x \sqsubseteq xy$. If $x \sqsubseteq z$ and x is maximal, then $xy = x = z$ for every y. The proof of item (ii) shows that if $x \sqsubseteq z$ and x is non-maximal then there is a (unique) y such that $xy = z$.

(ii) A rescaling is reversible iff the rescaling factor is non-zero, a translation is always reversible, and an element has diameter zero iff it is maximal.

(iii) The linear maps $p \mapsto \kappa_x p + \mu_x$ are either increasing order isomorphisms between the real line and itself (if $\kappa_x > 0$) or else constant maps (if $\kappa_x = 0$). $\square$

The above proof shows that the concatenation operation on partial numbers "multiplies diameters", in the sense that $\kappa_{xy} = \kappa_x \kappa_y$. Therefore $\kappa_{xy} \leq \kappa_x$ and $\kappa_{xy} \leq \kappa_y$, the equalities holding iff $x = \bot$ or $y = \bot$.

Concatenation of partial numbers has the following geometrical and computational interpretations. In a concatenation xy, the interval y refines the information given by x by selecting a subinterval of x. For example, the partial numbers $[0, 1]$, $[0, \frac{1}{2}]$, $[\frac{1}{2}, 1]$, and $[\frac{1}{3}, \frac{2}{3}]$ respectively select the whole interval, the first half, the second half, and the middle third part. Thus, concatenation of partial numbers allows for incremental computation on partial real numbers, also known as *lazy evaluation* (cf. Proposition 4). Hence the denomination *lazy* unit interval.

There is yet another geometrical interpretation of concatenation, induced by the isomorphism between the lazy unit interval and the unit triangle. The upper set of any non-maximal element x of the unit triangle is clearly a triangle, isomorphic to the unit triangle via a rescaling of the unit triangle followed by a translation. Thus, any element y can be interpreted either as an absolute address of a point in the unit triangle or else as a relative address of a point in the smaller triangle generated by x, namely the point xy, obtained by applying the same rescaling and translation to y.

We finish this subsection with the following lemma:

Lemma 3 *The bases of the lazy unit interval consisting of respectively all non-maximal partial numbers and all non-maximal partial numbers with rational* *end-points are submonoids of $(\mathcal{I}, \cdot, \perp)$ closed under existing quotients, in the sense that if b and c are basis elements with $b \sqsubseteq c$ then c/b is a basis element too.*

Proof. Existing quotients are given by

$$y/x = (y - \mu_x)/\kappa_x =_{\text{def}} [(\underline{y} - \mu_x)/\kappa_x, (\overline{y} - \mu_x)/\kappa_x)]$$

Therefore, if x and y have distinct (rational) end-points, so does y/x. $\square$

The basis consisting of all non-maximal partial numbers with dyadic end-points is a submonoid, but it is not closed under existing quotients.

3.4 Infinitely iterated concatenations

Let M be a monoid with a partial prefix order and joins of non-decreasing ω-chains, and let $\langle x_n \rangle_{n \geq 1}$ be a sequence of elements of M. Then we have that

$$x_1 \leq x_1 x_2 \leq \cdots \leq x_1 x_2 \cdots x_n \leq \cdots$$

The *infinitely iterated concatenation* of $\langle x_n \rangle_{n \geq 1}$ is defined to be the join of these partial concatenations, informally denoted by $x_1 x_2 \cdots x_n \cdots$. It is also convenient to use the informal notation $y_1 \sqcup y_2 \sqcup \cdots \sqcup y_n \sqcup \cdots$ for the join of a chain $y_1 \leq y_2 \leq \cdots \leq y_n \leq \cdots$.

An *interval expansion* of a partial real number x is a sequence of intervals $\langle x_n \rangle_{n \geq 1}$ such that $x = x_1 x_2 \cdots x_n \cdots$. For example, interval expansions formed from the intervals

$$\left[0, \frac{1}{10}\right], \left[\frac{1}{10}, \frac{2}{10}\right], \ldots, \left[\frac{8}{10}, \frac{9}{10}\right], \left[\frac{9}{10}, 1\right]$$

are essentially *decimal expansions.*

If we think of an infinitely iterated concatenation as a computation, the following proposition shows that we can compute in an incremental fashion if left translations are continuous:

Proposition 4 (Infinite associativity) *Let M be a monoid with infinitely iterated concatenations. Then M satisfies the ω-associativity law*

$$x_1(x_2 \cdots x_n \cdots) = x_1 x_2 \cdots x_n \cdots$$

iff left translations preserve joins of non-decreasing ω-chains.

Proof. ($\Rightarrow$) Assume that the ω-associativity law holds, let $\langle y_n \rangle_{n \geq 1}$ be a non-decreasing ω-chain of elements of M, and let $\langle x_n \rangle_{n \geq 1}$ be a sequence of elements of M such that $y_n x_n = y_{n+1}$. Then the join of the chain is the same as the infinitely iterated concatenation of the sequence $\langle x_n \rangle_{n \geq 1}$ with y_1 added as a new first element, because $y_1 x_1 x_2 \cdots x_n = y_n$, as an inductive argument shows. Therefore

$$\begin{aligned} a(y_1 \sqcup y_2 \sqcup \cdots \sqcup y_n \sqcup \cdots) &= a(y_1 x_1 x_2 \cdots x_n \cdots) \\ &= a y_1 x_1 x_2 \cdots x_n \cdots \\ &= a \sqcup a y_1 \sqcup a y_1 x_1 \sqcup \cdots \sqcup a y_1 x_1 \cdots x_n \sqcup \cdots \\ &= a \sqcup a y_1 \sqcup a y_2 \sqcup \cdots \sqcup a y_n \sqcup \cdots \\ &= a y_1 \sqcup a y_2 \sqcup \cdots \sqcup a y_n \sqcup \cdots \end{aligned}$$

($\Leftarrow$) Assume that left translations preserve joins of non-decreasing ω-chains. Then we have that

$$\begin{aligned} x_1(x_2 \cdots x_n \cdots) &= x_1(x_2 \sqcup x_2 x_3 \sqcup \cdots \sqcup x_2 x_3 \cdots x_n \sqcup \cdots) \\ &= x_1 x_2 \sqcup x_1 x_2 x_3 \sqcup \cdots \sqcup x_1 x_2 x_3 \cdots x_n \sqcup \cdots \\ &= x_1 \sqcup x_1 x_2 \sqcup x_1 x_2 x_3 \sqcup \cdots \sqcup x_1 x_2 x_3 \cdots x_n \sqcup \cdots \\ &= x_1 x_2 \cdots x_n \cdots \qquad \square \end{aligned}$$

We denote the infinitely iterated concatenation $xx \cdots x \cdots$ of a constant sequence with range x by x^ω. An immediate consequence of the above proposition is that x^ω is the least fixed point of cons_x for any monoid with infinite concatenations.

Now let $y_1 \leq y_2 \leq \cdots \leq y_n \leq \cdots$ be a chain of left cancelable elements. Then the sequence y_1, y_2/y_1, $y_3/y_2, \ldots,$ $y_{n+1}/y_n, \ldots$ has the elements of the chain as partial concatenations. Therefore the join of the chain is the same as the infinite concatenation of the induced sequence.

Proposition 5 *Consider the bases of $\mathcal{I}$ consisting of respectively all non-maximal partial numbers and all non-maximal partial numbers with rational end-points. Then there is a bijection between ω-*chains *of basis elements and* sequences *of basis elements, taking any chain to a sequence whose infinitely interated concatenation is the join of the chain.* $\square$

Therefore we can replace ω-chains of basis elements by (arbitrary) sequences of basis elements and work with infinitely iterated concatenations instead of joins.

For monoids with infinitely interated concatenations, it is natural to ask homomorphisms to preserve them.

Proposition 6 *A monoid homomorphism preserves infinitely iterated concatenations iff it preserves joins of increasing ω-chains*

Proof. Let $h : L \to M$ be a monoid homomorphism between monoids L and M.

($\Rightarrow$) Let $\langle y_n \rangle_{n \geq 1}$ be an non-decreasing ω-chain of elements of L having a join and let $\langle x_n \rangle_{n \geq 1}$ be a sequence of elements of L such that $y_n x_n = y_{n+1}$. Then we have that

$$\begin{aligned}
&h(y_1 \sqcup \cdots \sqcup y_n \sqcup \cdots) \\
&= h(y_1 x_1 \cdots x_n \cdots) \\
&= h(y_1)h(x_1) \cdots h(x_n) \cdots \\
&= h(y_1) \sqcup h(y_1)h(x_1) \sqcup \cdots \sqcup h(y_1)h(x_1) \cdots h(x_n) \sqcup \cdots \\
&= h(y_1) \sqcup \cdots \sqcup h(y_n) \sqcup \cdots
\end{aligned}$$

because $h(y_n)h(x_n) = h(y_{n+1})$.

($\Leftarrow$) Let $\langle x_i \rangle_{i \geq 1}$ be a sequence of elements of L having an infinitely iterated concatenation. Then we have that

$$\begin{aligned}
h\left(x_1 x_2 \cdots x_n \cdots\right) &= h\left(x_1 \sqcup x_1 x_2 \sqcup \cdots \sqcup x_1 x_2 \cdots x_n \sqcup \cdots\right) \\
&= h(x_1) \sqcup h(x_1 x_2) \sqcup \cdots \sqcup h(x_1 x_2 \cdots x_n) \sqcup \cdots \\
&= h(x_1) \sqcup h(x_1)h(x_2) \sqcup \cdots \sqcup h(x_1)h(x_2) \cdots h(x_n) \sqcup \cdots \\
&= h(x_1)h(x_2) \cdots h(x_n) \cdots \qquad \square
\end{aligned}$$

3.5 Continuous words

Concatenation of partial numbers in the lazy unit interval generalizes concatenation of words over any finite alphabet, in the following sense:

Proposition 7 *For every finite n-letter alphabet Σ, the monoid Σ^∞ is isomorphic to a submonoid of $\mathcal{I}$. Moreover, the induced embedding of Σ^∞ into $\mathcal{I}$ is a continuous monoid homomorphism.*

Proof. Without essential loss of generality, we prove the claim for the two-letter case $\Sigma = \{0, 2\}$. Let C be the submonoid of $\mathcal{I}$ finitely generated by the partial numbers $[0, 1/3]$ and $[2/3, 1]$. Then C clearly contains the partial numbers corresponding to the intervals successively produced in the construction of the Cantor set. Hence the joins of strictly increasing ω-chains of elements of C are the elements of the Cantor set. But these joins are the infinitely iterated concatenations of the generators. The set $\mathcal{C}$ of finite and infinite concatenations of the generators is also a submonoid of $\mathcal{I}$, which can be considered as the monoid of *partial Cantor numbers*. It is easy to see that $\{0, 2\}^\infty$ is isomorphic to $\mathcal{C}$. We know that $\{0, 2\}^*$ is the free monoid over $\{0, 2\}$. Hence there is a unique monoid homomorphism $h : \{0, 2\}^* \to \mathcal{C}$ such that $h(0) = [0, 1/3]$ and $h(2) = [2/3, 1]$. But h has a unique continuous extension to $\{0, 2\}^\infty$, because monoid homomorphisms are monotone and $\{0, 2\}^*$ consists of the finite elements (in the domain-theoretic sense) of $\{0, 2\}^\infty$. The resulting extension is a monoid homomorphism. Since it takes an infinite word x to the element of the Cantor set whose ternary expansion is x, it follows that h is a bijection, and therefore a monoid isomorphism. $\square$

Therefore, the elements of the lazy unit interval can be considered as "continuous words". For any word $x \in \Sigma^\infty$ let $\operatorname{length}(x) \in [0, \infty]$ denote the length of x defined in the usual way. For any continuous word $x \in \mathcal{I}$ and any real number $b > 1$ we define $\operatorname{length}_b(x) \in [0, \infty]$ to be $-\log_b(\kappa_x)$. Thus, $\operatorname{length}_b(x) = \infty$ iff x is maximal, and $\operatorname{length}_b(x) = 0$ iff x is bottom.

If a continuous word x is a partial realization of an unknown real number y, then $\operatorname{length}_b(x)$ is the number of correct digits of an expansion to base b of y that x allows us to know. For example, if we consider $x = [0.143, 0.145]$ as a partial realization of an unknown real number y then $\operatorname{length}_{10}(x)$ is roughly 2.7, meaning that x allows us to know two decimal digits of y.

Proposition 8 *Consider* $[0, \infty]$ *as a monoid under addition and as a domain under its prefix preorder (which coincides with its natural order).*

(i) For every real number $b > 1$, $\operatorname{length}_b : \mathcal{I} \to [0, \infty]$ *is a continuous monoid homomorphism.*
(ii) Let h *be the embedding of the monoid of words over a finite* n*-letter alphabet into* $\mathcal{I}$ *defined in Proposition 7. Then* $\operatorname{length} = \operatorname{length}_{n+1} \circ h$.

Proof. Routine verification. The first part follows from the fact that the concatenation operation multiplies diameters and that logarithms take multiplication to addition. $\square$

Corollary 9 *The infinitely iterated concatenation of a sequence of continuous words is maximal iff the sum of the lengths of the continuous words is* ∞.

Proof. The length function is a continuous monoid homomorphism and a partial number has infinite length iff it is maximal. □

The following proposition allows us to prove some properties of real number programs. We say that a map $f : \mathcal{I} \to \mathcal{I}$ is *guarded* if there is a real number $\delta > 0$ such that $\mathrm{length}(f(x)) \geq \mathrm{length}(x) + \delta$, called a *guarding constant* for f. Clearly, left translations cons_a with $a \neq \bot$ are guarded, with guarding constant $\mathrm{length}(a)$.

Proposition 10 *Any continuous guarded map $f : \mathcal{I} \to \mathcal{I}$ has a maximal partial number as its unique fixed point.*

Proof. Since $\mathrm{length}(f^n(\bot)) \geq n\delta$ for every n and length is a continuous homomorphism, $\mathrm{length}\left(\bigsqcup_n f^n(\bot)\right) \geq \sup_n n\delta = \infty$. This means that the least fixed point of f is maximal. Therefore it is the unique fixed point of f. □

3.6 Heads and tails of continuous words

We begin by considering the head and tail maps on words over the alphabet $\Sigma = \{0, 1\}$.

Let σ range over Σ, and define maps $\mathrm{tail} : \Sigma^\infty \to \Sigma^\infty$ and $\mathrm{head} : \Sigma^\infty \to \Sigma_\bot$

$$\begin{aligned} \mathrm{tail}(\epsilon) &= \epsilon \\ \mathrm{tail}(\sigma x) &= x \\ \mathrm{head}(\epsilon) &= \bot \\ \mathrm{head}(\sigma x) &= \sigma \end{aligned}$$

These maps are continuous, and we have that

$$\begin{aligned} \mathrm{tail}(\mathrm{cons}_\sigma(x)) &= x \\ \mathrm{head}(\mathrm{cons}_\sigma(x)) &= \sigma \end{aligned}$$

Recall that a pair of continuous maps $s : D \to E$ and $r : E \to D$ between dcpos D and E with the property that

$$r \circ s = \mathrm{id}$$

is called a *section-retraction pair* [3]. In this case s preserves all existing meets and joins, and $s \circ r$ is an idempotent.

By definition, tail is a retraction and cons_σ is a section. Recall also that if in addition the induced idempotent has

$$s \circ r \sqsubseteq \mathrm{id}$$

then the pair of maps s and r are said to be an *embedding-projection pair*. The pair of maps tail and cons_σ are not an embedding-projection pair, because for example

$$\mathrm{cons}_0(\mathrm{tail}(\epsilon)) = 0 \not\sqsubseteq \epsilon = \mathrm{id}(\epsilon)$$

Now define a continuous equality map $(x, y) \mapsto (x =_\perp y) : \Sigma_\perp \times \Sigma_\perp \to \{\mathrm{tt}, \mathrm{ff}\}_\perp$ by

$$(x =_\perp y) = \begin{cases} \mathrm{tt} \text{ if } x, y \in \Sigma \text{ and } x = y \\ \perp \text{ if } \perp \in \{x, y\} \\ \mathrm{ff} \text{ if } x, y \in \Sigma \text{ and } x \neq y \end{cases}$$

and a continuous conditional map

$$(p, x, y) \mapsto (\text{if } p \text{ then } x \text{ else } y) : \{\mathrm{tt}, \mathrm{ff}\}_\perp \times \Sigma^\infty \times \Sigma^\infty \to \Sigma^\infty$$

by

$$\text{if } p \text{ then } x \text{ else } y = \begin{cases} x \text{ if } p = \mathrm{tt} \\ \epsilon \text{ if } p = \perp \\ y \text{ if } p = \mathrm{ff} \end{cases}$$

Then we have that

$$x = \text{if } \mathrm{head}(x) =_\perp 0 \text{ then } \mathrm{cons}_0(\mathrm{tail}(x)) \text{ else } \mathrm{cons}_1(\mathrm{tail}(x)) \tag{1}$$

Manipulations of this equation lead to usual explicit and recursive definitions of several functions. For example, it follows from (1) that the identity function satisfies the equation

$$\mathrm{id}(x) = \text{if } \mathrm{head}(x) =_\perp 0 \text{ then } \mathrm{cons}_0(\mathrm{id}(\mathrm{tail}(x))) \text{ else } \mathrm{cons}_1(\mathrm{id}(\mathrm{tail}(x)))$$

In fact, the identity function is the unique continuous map which satisfies the functional equation

$$f(x) = \text{if } \mathrm{head}(x) =_\perp 0 \text{ then } \mathrm{cons}_0(f(\mathrm{tail}(x))) \text{ else } \mathrm{cons}_1(f(\mathrm{tail}(x)))$$

The function that switches the rôles of the letters 0 and 1 can be given by the the recursive definition

$$g(x) = \text{if } \mathrm{head}(x) =_\perp 0 \text{ then } \mathrm{cons}_1(g(\mathrm{tail}(x))) \text{ else } \mathrm{cons}_0(g(\mathrm{tail}(x)))$$

In this subsection we look for a generalization of this theme for partial numbers. The first problem to be overcome is that continuous words have non-integral lengths and in particular non-zero lengths < 1. Moreover, a continuous word of length > 1 has infinitely many prefixes of length 1. Since the head map on words extracts a prefix of length 1, it is not immediately clear what the head map on continuous words should be for words of length < 1.

Under the embedding of Σ^∞ into $\mathcal{I}$ defined in Proposition 7, if we put

$l = [0, 1/3]$ and $r = [2/3, 1]$

then Equation (1) becomes

$$x = \text{if } x <_\perp \tfrac{1}{2} \text{ then } \mathrm{cons}_l(x/l) \text{ else } \mathrm{cons}_r(x/r) \qquad (1')$$

where the conditional map is defined in the same way for partial numbers and the continuous comparison map $(x, y) \mapsto (x <_\perp y) : \mathcal{I} \times \mathcal{I} \to \{\mathrm{tt}, \mathrm{ff}\}_\perp$ is defined by

$$(x <_\perp y) = \begin{cases} \mathrm{tt} \text{ if } x < y \\ \perp \text{ if } x \simeq y \\ \mathrm{ff} \text{ if } x > y \end{cases}$$

Remark 11 *If we had $x <_\perp x = \mathrm{ff}$ then this map would not be continuous.* $\square$

Then we need a continuous map $\mathrm{tail}_a : \mathcal{I} \to \mathcal{I}$, for any left cancelable partial number a, with the property that

$\mathrm{tail}_a(x) = x/a$ for every x with $x \sqsubseteq a$, that is, with the property that tail_a is a left-inverse of cons_a. A naïve attempt to obtain such a map would be to put $\mathrm{tail}_a(x) = \bot$ if $x \not\sqsubseteq a$. But then tail_a would not be continuous.

Lemma 12 *Let $a \in \mathcal{I}$ be a non-maximal element, let $\mathrm{tail}_a : \mathcal{I} \to \mathcal{I}$ be* any *continuous left inverse of cons_a, and let f_a be the induced idempotent. Then for every $x \in \mathcal{I}$,*

(i) $a \sqsubseteq f_a(x)$
(ii) $f_a(x) = x$ *iff* $a \sqsubseteq x$
(iii) $\mathrm{tail}_a(x) = f_a(x)/a$

Proof. (1) $a \sqsubseteq \mathrm{cons}_a(\mathrm{tail}_a(x)) = f_a(x)$.

(2) ($\Rightarrow$) Immediate consequence of (1) above. ($\Leftarrow$) If $a \sqsubseteq x$ then $x = a(x/a)$ and $f_a(x) = \mathrm{cons}_a(\mathrm{tail}_a(a(x/a))) = \mathrm{cons}_a(x/a) = x$.

(3) $\mathrm{tail}_a(x) = \mathrm{cons}_a(\mathrm{tail}_a(x))/a = f_a(x)/a$. □

Thus, in order to obtain a continuous left inverse of cons_a we first look for a continuous map f_a enjoying properties (1) and (2) above. Assume that $x \simeq a$. Then $f_a(x) \sqsubseteq f_a(x \sqcup a) = x \sqcup a$, because $x \sqsubseteq x \sqcup a$. Since $a \sqsubseteq x \sqcup a$, by (1) and (2) it is natural to let $f_a(x)$ be $x \sqcup a$. Then $f_a(x)$ can be thought as the truncation of x to a, because $x \sqcup a$ is the greatest subinterval of x contained in a.

Lemma 13 *Let a be an element of $\mathcal{I}$. If $\mathrm{join}_a : \mathcal{I} \to \mathcal{I}$ is a monotone map such that*

$\mathrm{join}_a(x) = x \sqcup a$ *for all* $x \simeq a$.

then $\mathrm{join}_a(x) = \underline{a}$ *for all* $x < a$ *and* $\mathrm{join}_a(x) = \overline{a}$ *for all* $x > a$.

Proof. Here we write $\underline{a}$ to mean $[\underline{a}, \underline{a}]$, of course. Assume that $x < a$. This means that $\overline{x} < \underline{a}$. Then $[\underline{x}, \underline{a}] \sqsubseteq x$. Hence $\underline{a} = \mathrm{join}_a([\underline{x}, \underline{a}]) \sqsubseteq \mathrm{join}_a(x)$. Therefore $\underline{a} = \mathrm{join}_a(x)$, because $\underline{a}$ is maximal. The other case is similar. □

Lemma 14 *For all non-maximal $a \in \mathcal{I}$ define maps $\mathrm{join}_a, \mathrm{tail}_a : \mathcal{I} \to \mathcal{I}$ by*

$$\mathrm{join}_a(x) = \begin{cases} \underline{a} & \textit{if } x < a \\ x \sqcup a & \textit{if } x \simeq a \\ \overline{a} & \textit{if } x > a \end{cases}$$
$$\mathrm{tail}_a(x) = \mathrm{join}_a(x)/a$$

Then join_a *and* tail_a *are continuous.*

Proof.

$$\mathrm{join}_a(x) = \max(\underline{a}, \min(x, \overline{a})) =_{\mathrm{def}} [\max(\underline{a}, \min(\underline{x}, \overline{a})),\ \max(\underline{a}, \min(\overline{x}, \overline{a}))]$$

Hence

$$\begin{aligned} \mathrm{tail}_a(x) &= \max(\underline{a}, \min(x, \overline{a}))/a \ = \ (\max(\underline{a}, \min(x, \overline{a})) - \mu_a)/\kappa_a \\ &= \max(0, \min((x - \mu_a)/\kappa_a,\ 1)) \end{aligned}$$

Since join_a and tail_a are the pointwise extensions of the non-decreasing continuous maps

$$\begin{aligned} &r \mapsto \max(\underline{a}, \min(r, \overline{a})) :\ [0,1] \to [0,1] \\ &r \mapsto \max(0, \min((r - \mu_a)/\kappa_a,\ 1)) :\ [0,1] \to [0,1] \end{aligned}$$

respectively, it follows that they are continuous (see Moore [25]). □

Proposition 15 *For all non-maximal* $a \in \mathcal{I}$*, the maps* cons_a *and* tail_a *form a section-retraction pair, with* join_a *as the induced idempotent.*

Proof. Immediate consequences of Lemmas 12 and 14. □

Notice that the definition of join_a makes sense even for maximal partial numbers a; in this case join_a is a constant map with range a. If we define $x \,\Box\, y = \mathrm{join}_x(y)$, then $(\mathcal{I}, \Box, \perp)$ is a monoid whose prefix preorder coincides with the information order on $\mathcal{I}$. Of course, this monoid is not isomorphic to the monoid $(\mathcal{I}, \cdot, \perp)$ This shows that there are several essentially different ways of making $\mathcal{I}$ into a monoid inducing the information order of $\mathcal{I}$.

The second problem to be overcome is that Equation (1′) is true only for a restricted set of partial numbers x. This problem is partially solved by replacing l and r by

$$L = [0, \tfrac{1}{2}] \text{ and } R = [\tfrac{1}{2}, 1]$$

respectively. By replacing the (sequential) conditional by the parallel one, namely $(p, x, y) \mapsto (\text{pif } p \text{ then } x \text{ else } y) : \{\mathrm{tt}, \mathrm{ff}\}_\perp \times \mathcal{I} \times \mathcal{I} \to \mathcal{I}$ defined by

$$\text{pif } p \text{ then } x \text{ else } y = \begin{cases} x & \text{if } p = \mathrm{tt} \\ x \sqcap y & \text{if } p = \perp \\ y & \text{if } p = \mathrm{ff} \end{cases}$$

(cf. [28] and Subsection 4.5), the problem is completely solved. Recall that the meet $x \sqcap y$ is the greatest common prefix of the continuous words x and y.

Finally, for each $r \in [0, 1]$ define $\mathrm{head}_r : \mathcal{I} \to \{\mathrm{tt}, \mathrm{ff}\}_\perp$ by

$$\mathrm{head}_r(x) = (x <_\perp r)$$

Theorem 16 *For any partial number $x \in \mathcal{I}$,*

$$x = \text{pif } \mathrm{head}_{\frac{1}{2}}(x) \text{ then } \mathrm{cons}_L(\mathrm{tail}_L(x)) \text{ else } \mathrm{cons}_R(\mathrm{tail}_R(x)) \qquad (1'')$$

Proof. By virtue of Proposition 15, it suffices to check the crucial case $x \simeq \frac{1}{2}$. Equation (1″) is true in this case because

$$\mathrm{join}_L(x) = [\underline{x}, \tfrac{1}{2}] \quad \mathrm{join}_R(x) = [\tfrac{1}{2}, \overline{x}]$$

and therefore $\mathrm{join}_L(x) \sqcap \mathrm{join}_R(x) = x$. □

Of course, there is nothing special about the number $\frac{1}{2}$ and the partial numbers $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$; the above theorem works for any binary partition of the unit interval, and can be extended to any finite n-ary partition with $n \geq 2$.

The idea behind the above theorem is strongly related to the ideas presented in [14]; see Examples 18, 19, and 25. See also Smyth's constructions of the unit interval as inverse limits of finite topological structures [36].

Example 17 (Path composition) *The parallel conditional can be used to overcome the fact that $x <_\perp x$ is $\perp$ instead of* ff.

Let E be any domain, and let $f, g : \mathcal{I} \to E$ be continuous maps. Then f and g are (generalized) paths in the domain E. (The restrictions of f and g to the singleton intervals are paths in the usual sense [20].) If $f(1) = g(0)$ the paths are said to be composable. *Now define a continuous map $h : \mathcal{I} \to E$ by*

$$h(x) = \text{pif } x <_\perp \tfrac{1}{2} \text{ then } f(2x) \text{ else } g(2x-1)$$

If the paths are composable, then h is a (generalized) composite path. *Let us check the crucial case $x = \frac{1}{2}$:*

$$\begin{aligned} h(\tfrac{1}{2}) &= \text{pif } \tfrac{1}{2} <_\perp \tfrac{1}{2} \text{ then } f(2\tfrac{1}{2}) \text{ else } g(2\tfrac{1}{2}-1) \;=\; \text{pif } \perp \text{ then } f(1) \text{ else } g(0) \\ &= f(1) \sqcap g(0) \;=\; f(1) \;=\; g(0) \end{aligned}$$

Notice that the sequential conditional would produce $\perp$ instead of $f(1) \sqcap g(0)$, and therefore h would be undefined at $\frac{1}{2}$.

If f and g are not composable, then h is a generalized composite path with a jump *at $\frac{1}{2}$, namely $f(1) \sqcap g(0)$. For instance, if $E = \mathcal{I}$ and f and g are constant maps with range 0 and 1, then h can be thought as a* switch *which turns on at time $\frac{1}{2}$. In this case, the switch is in the transition state $[0,1] = 0 \sqcap 1$ at time $\frac{1}{2}$. Notice that even in this case h* is *Scott continuous, because it is a composition of continuous maps.* □

The following recursive definitions generalize the recursive definition for real numbers presented in [14] to partial real numbers.

Example 18 (Complement) *We now derive a recursive definition of the extension of the complement map $r \mapsto 1 - r$ on $[0,1]$ to $\mathcal{I}$ given by*

$$1 - x = [1 - \overline{x},\, 1 - \underline{x}]$$

Of course, for maximal partial numbers, this indeed coincides with the original map, up to the identification of real numbers and maximal partial numbers.

We begin with the following observation:

(i) If $x \subseteq L$ then $1 - x \subseteq R$.
(ii) If $x \subseteq R$ then $1 - x \subseteq L$.

This leads one to write down the following set of incomplete equations:

$$\begin{aligned} 1 - \text{cons}_L(x) &= 1 - \text{cons}_R(\cdots) \\ 1 - \text{cons}_R(x) &= 1 - \text{cons}_L(\cdots) \end{aligned}$$

If we can fill the gaps with expressions depending on $1-x$ then we are almost done. Routine algebra shows that the gaps can be filled in an essentially unique way:

$$\begin{aligned} 1-\mathrm{cons}_L(x) &= 1-\mathrm{cons}_R(1-x) \\ 1-\mathrm{cons}_R(x) &= 1-\mathrm{cons}_L(1-x) \end{aligned}$$

Now we reduce this two equations to a single equation, by means of the primitive operations that we have introduced:

$$1-x = \mathrm{pif}\ \mathrm{head}_{\frac{1}{2}}(x)\ \mathrm{then}\ \mathrm{cons}_R(1-\mathrm{tail}_L(x))\ \mathrm{else}\ \mathrm{cons}_L(1-\mathrm{tail}_R(x))$$

This can be considered as a recursive definition of complement, because if we eliminate recursion by means of the least fixed point operator then it is possible to prove by fixed point induction [3] that complement is the least (in fact, the unique) fixed point of the induced functional. We use fixed point induction in conjunction with an induction principle for partial real numbers generalizing the one introduced in [14].

It is illustrative to unfold $1-[a,b]$ for $[a,b] \in \mathcal{I}$ with $1/2 \in [a,b]$:

$$\begin{aligned} 1-[a,b] &= \mathrm{pif}\ \perp\ \mathrm{then}\ \mathrm{cons}_R(1-[2a,1])\ \mathrm{else}\ \mathrm{cons}_L(1-[0,2b-1]) \\ &= \mathrm{cons}_R([0,1-2a]) \sqcap \mathrm{cons}_L([2-2b,1]) \\ &= [\tfrac{1}{2},1-a] \sqcap [1-b,\tfrac{1}{2}] \ = \ [1-b,1-a] \end{aligned}$$

Notice that the above recursive definition resembles a recursive definition with respect to binary representation, with L and R corresponding to the digits 0 and 1 respectively. $\square$

Example 19 (Average) *We now derive a recursive definition of the extension of the average operation*

$$r \oplus s = \frac{r+s}{2}$$

given by

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

The derivation follows the same pattern as the derivation of the recursive definition of complement given in the above example.

Initial observation:

(i) If $x, y \subseteq L$ then $x + y \subseteq L$.
(ii) If $x \subseteq L$ and $y \subseteq R$ then $x + y \subseteq C$.
(iii) If $x, y \subseteq R$ then $x + y \subseteq R$.

where $C = [1/4,\ 3/4]$.

Incomplete equations:

$$\begin{aligned}
\mathrm{cons}_L(x) \oplus \mathrm{cons}_L(y) &= \mathrm{cons}_L(\cdots)\\
\mathrm{cons}_L(x) \oplus \mathrm{cons}_R(y) &= \mathrm{cons}_C(\cdots)\\
\mathrm{cons}_R(x) \oplus \mathrm{cons}_L(y) &= \mathrm{cons}_C(\cdots)\\
\mathrm{cons}_R(x) \oplus \mathrm{cons}_R(y) &= \mathrm{cons}_R(\cdots)
\end{aligned}$$

Gaps filled:

$$\begin{aligned}
\mathrm{cons}_L(x) \oplus \mathrm{cons}_L(y) &= \mathrm{cons}_L(x \oplus y)\\
\mathrm{cons}_L(x) \oplus \mathrm{cons}_R(y) &= \mathrm{cons}_C(x \oplus y)\\
\mathrm{cons}_R(x) \oplus \mathrm{cons}_L(y) &= \mathrm{cons}_C(x \oplus y)\\
\mathrm{cons}_R(x) \oplus \mathrm{cons}_R(y) &= \mathrm{cons}_R(x \oplus y)
\end{aligned}$$

Reduction to a single equation:

$$\begin{array}{lll}
x \oplus y = \mathrm{pif}\ \mathrm{head}_{\frac{1}{2}}(x)\ \mathrm{then}\ \mathrm{pif}\ \mathrm{head}_{\frac{1}{2}}(y) & \mathrm{then} & \mathrm{cons}_L(\mathrm{tail}_L(x) \oplus \mathrm{tail}_L(y))\\
 & \mathrm{else} & \mathrm{cons}_C(\mathrm{tail}_L(x) \oplus \mathrm{tail}_R(y))\\
\qquad\qquad\qquad\qquad\quad \mathrm{else}\ \ \mathrm{pif}\ \mathrm{head}_{\frac{1}{2}}(y) & \mathrm{then} & \mathrm{cons}_C(\mathrm{tail}_R(x) \oplus \mathrm{tail}_L(y))\\
 & \mathrm{else} & \mathrm{cons}_R(\mathrm{tail}_R(x) \oplus \mathrm{tail}_R(y)),
\end{array}$$

In practice it may be convenient to use the more familiar notations

$x < \frac{1}{2}$, $x/2$, $x/2 + 1/4$, $x/2 + \frac{1}{2}$, $\min(2x,\ 1)$, *and* $\max(0,\ 2x - 1)$

instead of

$\mathrm{head}_{\frac{1}{2}}(x)$, $\mathrm{cons}_L(x)$, $\mathrm{cons}_C(x)$, $\mathrm{cons}_R(x)$, $\mathrm{tail}_L(x)$, *and* $\mathrm{tail}_R(x)$

respectively, and even write $2x$ and $2x - 1$ for the last two cases by an abuse of notation. With this notation the above recursive definition becomes

$$\begin{array}{lll}
x \oplus y = \mathrm{pif}\ x < \frac{1}{2}\ \mathrm{then}\ \mathrm{pif}\ y < \frac{1}{2} & \mathrm{then} & (2x \oplus 2y)/2\\
 & \mathrm{else} & (2x \oplus (2y - 1))/2 + 1/4\\
\qquad\qquad\qquad\quad \mathrm{else}\ \mathrm{pif}\ y < \frac{1}{2} & \mathrm{then} & ((2x - 1) \oplus 2y)/2 + 1/4
\end{array}$$

$$\text{else} \quad ((2x-1) \oplus (2y-1))/2 + \tfrac{1}{2} \quad \square$$

Example 20 (Multiplication) *We now derive a recursive definition of multiplication, which is extended in the same way as average. During this example juxtaposition means multiplication instead of concatenation. Again, the derivation follows the same pattern.*

Initial observation:

(i) *If* $x, y \subseteq L$ *then* $xy \subseteq [0,\ 1/4]$.
(ii) *If* $x \subseteq L$ *and* $y \subseteq R$ *then* $xy \subseteq [0,\ 1/2]$.
(iii) *If* $x, y \subseteq R$ *then* $xy \subseteq [1/4,\ 1]$.

Incomplete equations:

$$\begin{aligned}
\mathrm{cons}_L(x) \times \mathrm{cons}_L(y) &= \mathrm{cons}_{[0,\,1/4]}(\cdots)\\
\mathrm{cons}_L(x) \times \mathrm{cons}_R(y) &= \mathrm{cons}_{[0,\,1/2]}(\cdots)\\
\mathrm{cons}_R(x) \times \mathrm{cons}_L(y) &= \mathrm{cons}_{[0,\,1/2]}(\cdots)\\
\mathrm{cons}_R(x) \times \mathrm{cons}_R(y) &= \mathrm{cons}_{[1/4,\,1]}(\cdots)
\end{aligned}$$

Gaps filled with expressions depending on x, y *and* xy*:*

$$\begin{aligned}
\mathrm{cons}_L(x) \times \mathrm{cons}_L(y) &= \mathrm{cons}_{[0,\,1/4]}(xy)\\
\mathrm{cons}_L(x) \times \mathrm{cons}_R(y) &= \mathrm{cons}_{[0,\,1/2]}\left(\frac{x+xy}{2}\right)\\
\mathrm{cons}_R(x) \times \mathrm{cons}_L(y) &= \mathrm{cons}_{[0,\,1/2]}\left(\frac{xy+y}{2}\right)\\
\mathrm{cons}_R(x) \times \mathrm{cons}_R(y) &= \mathrm{cons}_{[1/4,\,1]}\left(\frac{x+xy+y}{3}\right)
\end{aligned}$$

Now we have to find a recursive definition of ternary average. In this case there are eight cases to consider, but since the operation is permutative (permutativity is the generalization of commutativity to n-ary operations), there are really only four cases to consider. We omit the routine derivation and the reduction to a single equation. $\square$

3.7 Computation rules for continuous words

The following lemma shows how to reduce expressions e denoting non-bottom partial numbers to expressions of the form $\mathrm{cons}_a(e')$ with $a \neq \bot$. Such an expression is called a *head-normal form.* The idea is that if an expression e has a head-normal form $\mathrm{cons}_a(e')$ then we know that its value is contained in a.

Thus, a head-normal form is a *partially evaluated* expression. A better partial evaluation of e is obtained by partially evaluating e', obtaining a head-normal form $\mathrm{cons}_b(e'')$, and applying rule (i) below to $\mathrm{cons}_a(\mathrm{cons}_b(e''))$ in order to obtain the more informative head-normal form $\mathrm{cons}_{ab}(e'')$ of e, and so on.

Lemma 21 *For all non-maximal $a, b \in \mathcal{I}$ and all $x \in \mathcal{I}$,*

(i) $\mathrm{cons}_a(\mathrm{cons}_b(x)) = \mathrm{cons}_{ab}(x)$
(ii) $\mathrm{tail}_a(\mathrm{cons}_b(x)) = L^\omega$ *if* $b \leq a$
(iii) $\mathrm{tail}_a(\mathrm{cons}_b(x)) = R^\omega$ *if* $b \geq a$
(iv) $\mathrm{tail}_a(\mathrm{cons}_b(x)) = \mathrm{cons}_{b/a}(x)$ *if* $a \sqsubseteq b$
(v) $\mathrm{tail}_a(\mathrm{cons}_b(x)) = \mathrm{cons}_{(a \sqcup b)/a}(\mathrm{tail}_{(a \sqcup b)/b}(x))$ *if* $a \simeq b$
(vi) $\mathrm{head}_r(\mathrm{cons}_a(x)) = \mathrm{tt}$ *if* $a < r$
(vii) $\mathrm{head}_r(\mathrm{cons}_a(x)) = \mathrm{ff}$ *if* $a > r$
(viii) $\mathrm{pif}\ \mathrm{tt}\ \mathrm{then}\ x\ \mathrm{else}\ y = x$
(ix) $\mathrm{pif}\ \mathrm{ff}\ \mathrm{then}\ x\ \mathrm{else}\ y = y$
(x) $\mathrm{pif}\ p\ \mathrm{then}\ \mathrm{cons}_a(x)\ \mathrm{else}\ \mathrm{cons}_b(y) =$

$$\mathrm{cons}_{a \sqcap b}(\mathrm{pif}\ p\ \ \begin{array}{l}\mathrm{then}\ \ \mathrm{cons}_{a/(a \sqcap b)}(x) \\ \mathrm{else}\ \ \mathrm{cons}_{b/(a \sqcap b)}(y))\end{array}$$

Proof. (i) This is the associativity law expressed in terms of left translations. (ii) If $b \leq a$ then $\mathrm{tail}_a(bx) = 0 = [0, \frac{1}{2}]^\omega$. (iii) Similar.

(iv) Equivalent to the fact that $(bx)/a = (b/a)x$.

(v) For all $c \simeq x$, $\mathrm{cons}_b(\mathrm{join}_c(x)) = \mathrm{cons}_b(c \sqcup x) = \mathrm{cons}_b(c) \sqcup \mathrm{cons}_b(x) = \mathrm{join}_{bc}(\mathrm{cons}_b(x))$, because left-translations preserve existing joins. Hence

$$\begin{aligned}\mathrm{tail}_a(\mathrm{cons}_b(x)) &= \mathrm{join}_a(\mathrm{cons}_b(x))/a \ = \ \mathrm{join}_{a \sqcup b}(\mathrm{cons}_b(x))/a \\ &= \mathrm{join}_{b((a \sqcup b)/b)}(\mathrm{cons}_b(x))/a \ = \ \mathrm{cons}_b(\mathrm{join}_{(a \sqcup b)/b}(x))/a \\ &= \mathrm{cons}_b(\mathrm{cons}_{(a \sqcup b)/b}(\mathrm{tail}_{(a \sqcup b)/b}(x)))/a \\ &= \mathrm{cons}_{a \sqcup b}(\mathrm{tail}_{(a \sqcup b)/b}(x))/a \ = \ \mathrm{cons}_{(a \sqcup b)/a}(\mathrm{tail}_{(a \sqcup b)/b}(x))\end{aligned}$$

The second step follows from the fact that $\mathrm{join}_a(y) = \mathrm{join}_{a \sqcup b}(y)$ if $a \simeq b$ and $b \sqsubseteq y$. The last step follows from the fact that $(cy)/a = (c/a)y$.

(vi–ix) Immediate. (x) It suffices to show that

$$\mathrm{cons}_a(x) \sqcap \mathrm{cons}_b(y) = \mathrm{cons}_{a \sqcap b}(\mathrm{cons}_{a/(a \sqcap b)}(x) \sqcap \mathrm{cons}_{b/(a \sqcap b)}(y))$$

Since left translations preserve meets and $a = c(a/c)$ for all c, the result follows by taking $c = a \sqcap b$. $\square$

3.8 The lazy unit interval acting on the lazy real line

In this subsection we extend the results of the previous subsections from the lazy unit interval to the lazy real line.

The concatenation operation

$$xy = \kappa_x y + \mu_x$$

originally defined for x and y in the lazy unit interval makes sense for x and y ranging over the lazy real line. With this extension, $\mathcal{R}$ becomes a monoid too. But its prefix preorder does not coincide with its information order, and it is due to this reason that we initially restricted ourselves to the lazy unit interval. In fact, if $y = ax$ this does not mean that y is contained in a. However, if we know that x is in the lazy unit interval then $y = ax$ do imply that y is contained in a, even if y is not in the lazy unit interval. This is the content of the following theorem.

Theorem 22 *The map $(x, y) \mapsto xy : \mathcal{R} \times \mathcal{I} \to \mathcal{R}$ is a (right) action of the monoid $(\mathcal{I}, \cdot, \bot)$ on the monoid $(\mathcal{R}, \cdot, [0, 1])$, inducing the information order of $\mathcal{R}$:*

(i) For all $x \in \mathcal{R}$ and all $y, z \in \mathcal{I}$, $x\bot = x$ and $(xy)z = x(yz)$.
(ii) For all $x, z \in \mathcal{R}$, $x \sqsubseteq z$ iff $xy = z$ for some $y \in \mathcal{I}$, such a y being unique iff x is non-maximal.

Moreover, for all $a \in \mathcal{R}$, the map $\mathrm{ricons}_a : \mathcal{I} \to \mathcal{R}_\bot$ defined by $\mathrm{ricons}_a(x) = ax$ preserves all meets and all existing joins. $\square$

Given $x, z \in \mathcal{R}$ with $x \sqsubseteq z$ and x non-maximal, denote the unique $y \in \mathcal{I}$ such that $xy = z$ by z/x.

For each non-maximal $a \in \mathcal{R}$, define a strict continuous map $\mathrm{irtail}_a : \mathcal{R}_\bot \to \mathcal{I}$ and a continuous non-strict map $\mathrm{rrjoin}_a : \mathcal{R}_\bot \to \mathcal{R}_\bot$ by extending Lemma 14 in the obvious way, so that

$$\mathrm{irtail}_a(x) = x/a \text{ if } x \sqsubseteq a$$

Finally, for each non-maximal $a \in \mathcal{R}$ define a strict continuous map $\mathrm{rrcons}_a : \mathcal{R}_\bot \to \mathcal{R}_\bot$ by

$$\mathrm{rrcons}_a(x) = ax$$

Proposition 23 *For all non-maximal $a \in \mathcal{R}$, the maps ricons_a and irtail_a form a section-retraction pair, with rrjoin_a as the induced idempotent.* $\square$

Thus, $\mathcal{I}$ is a retract of $\mathcal{R}_\perp$, in many ways. In particular,

(i) $\mathrm{ricons}_{[0,1]} : \mathcal{I} \to \mathcal{R}_\perp$ is the inclusion,
(ii) $\mathrm{irtail}_{[0,1]} : \mathcal{R}_\perp \to \mathcal{I}$ is a "truncated projection", and
(iii) $\mathrm{rrjoin}_{[0,1]} : \mathcal{R}_\perp \to \mathcal{R}_\perp$ is the co-extension of the truncated projection.

Example 24 (Addition) *We obtain a recursive definition of addition in $\mathcal{R}_\perp$ by reducing addition to the average operation on $\mathcal{I}$ defined in Example 19. First notice that*

$$\begin{aligned} x + y = {} & \mathrm{pif}\ x < 0 \vee y < 0 \ \mathrm{then}\ 2\left(\tfrac{x+1}{2} + \tfrac{y+1}{2}\right) - 2 \\ \mathrm{else}\quad & \mathrm{pif}\ x > 1 \vee y > 1 \ \mathrm{then}\ 2\left(\tfrac{x}{2} + \tfrac{y}{2}\right) \\ & \qquad\qquad\qquad\quad \mathrm{else}\ \ 2\left(\tfrac{x+y}{2}\right) \end{aligned}$$

where $\vee$ is "parallel or" [28]. Hence

$$\begin{aligned} x + y = {} & \mathrm{pif}\ x < 0 \vee y < 0 \ \mathrm{then}\ 2\left(\tfrac{x+1}{2} + \tfrac{y+1}{2}\right) - 2 \\ \mathrm{else}\quad & \mathrm{pif}\ x > 1 \vee y > 1 \ \mathrm{then}\ 2\left(\tfrac{x}{2} + \tfrac{y}{2}\right) \\ & \qquad\qquad\qquad\quad \mathrm{else}\ \ 2\,\mathrm{out}(\mathrm{in}(x) \oplus \mathrm{in}(y)) \end{aligned}$$

where $\mathrm{in} = \mathrm{irtail}_{[0,1]} : \mathcal{R}_\perp \to \mathcal{I}$ *and* $\mathrm{out} = \mathrm{ricons}_{[0,1]} : \mathcal{I} \to \mathcal{R}_\perp$.

The operation out *is of fundamental importance in this recursive definition; it is this operation that makes the above recursive definition "get off the ground". More formally, this is a non-strict operation, and some non-strict operation is needed if the fixed point operator (used to solve the above equation) is to produce a non-bottom solution. If we want to avoid using intermediate maps on the unit interval in recursive definitions of maps on the real line, we can use the non-strict operation* rrjoin *to get off the ground. See example below.* □

Multiplication on $\mathcal{R}_\perp$ can be obtained in a similar way from multiplication on $\mathcal{I}$.

Example 25 (Logarithm) *The following recursive definition of the logarithm function* $\log_b$, *with $b > 1$, is based on [14]. We first reduce the calculation of* $\log_b(x)$ *for x arbitrary to a calculation of* $\log_b(x)$ *for $x \subseteq [1, b]$, so that* $\log_b(x) \subseteq [0, 1]$, *and then recursively consider the cases $x = \sqrt{x'}$ and $x = \sqrt{bx''}$. Notice that such x' and x'' are again contained in $[1, b]$, and that both cases hold iff $x = \sqrt{b}$ iff $x^2 = b$. Also, the reduction process diverges if x contains non-positive numbers.*

$$\begin{aligned} \log_b(x) = {} & \mathrm{pif}\ x > b \ \mathrm{then} \log_b(x/b) + 1 \\ \mathrm{else}\ \ & \mathrm{pif}\ x < 1 \ \mathrm{then} \log_b(bx) - 1 \\ \mathrm{else}\ \ & \mathrm{rrjoin}_{[0,1]}\left(\mathrm{pif}\ x^2 < b \ \mathrm{then}\ \tfrac{\log_b(x^2)}{2} \ \mathrm{else}\ \tfrac{\log_b(x^2/b)+1}{2}\right) \end{aligned}$$

Recall that $\mathrm{rrjoin}_{[0,1]}(x) = \max(0, \min(x, 1))$. *The application of the map* rrjoin *therefore seems redundant, because the tests preceding the recursive call of the logarithm function in the argument of* rrjoin *would ensure that the result is contained in* $[0, 1]$. *But the logarithm function is what the recursive definition is defining and therefore* it *has to ensure that this is indeed the case. As explained in Example 24, such a map is needed to "get off the ground".* □

The head-normal forms for $\mathcal{R}_\perp$ are taken as the expressions of the form $\mathrm{ricons}_a(e)$.

Lemma 26 *For all non-maximal $a \in \mathcal{R}$ and $b \in \mathcal{I}$, and all $x \in \mathcal{I}$,*

(i) (a) $\mathrm{rrcons}_a(\mathrm{rrcons}_b(x)) = \mathrm{rrcons}_{ab}(x)$
(b) $\mathrm{rrcons}_a(\mathrm{ricons}_b(x)) = \mathrm{ricons}_{ab}(x)$
(c) $\mathrm{ricons}_a(\mathrm{cons}_b(x)) = \mathrm{ricons}_{ab}(x)$
(ii) $\mathrm{irtail}_a(\mathrm{ricons}_b(x)) = L^\omega$ *if* $b \leq a$
(iii) $\mathrm{irtail}_a(\mathrm{ricons}_b(x)) = R^\omega$ *if* $b \geq a$
(iv) $\mathrm{irtail}_a(\mathrm{ricons}_b(x)) = \mathrm{cons}_{b/a}(x)$ *if* $a \sqsubseteq b$
(v) $\mathrm{irtail}_a(\mathrm{ricons}_b(x)) = \mathrm{cons}_{(a\sqcup b)/a}(\mathrm{tail}_{(a\sqcup b)/b}(x))$ *if* $a \simeq b$
(vi) $\mathrm{rhead}_r(\mathrm{ricons}_a(x)) = \mathrm{tt}$ *if* $a < r$
(vii) $\mathrm{rhead}_r(\mathrm{ricons}_a(x)) = \mathrm{ff}$ *if* $a > r$
(viii) $\mathrm{pif\ tt\ then}\ x\ \mathrm{else}\ y = x$
(ix) $\mathrm{pif\ ff\ then}\ x\ \mathrm{else}\ y = y$
(x) $\mathrm{pif}\ p\ \mathrm{then}\ \mathrm{ricons}_a(x)\ \mathrm{else}\ \mathrm{ricons}_b(y) =$

$$\mathrm{ricons}_{a\sqcap b}(\mathrm{pif}\ p\ \begin{array}{ll}\mathrm{then} & \mathrm{ricons}_{a/(a\sqcap b)}(x)\\ \mathrm{else} & \mathrm{ricons}_{b/(a\sqcap b)}(y)\end{array}$$

where the parallel conditional, the comparison map, and the head map for $\mathcal{R}_\perp$ are defined in the same way as for $\mathcal{I}$. Notice that the above lemma is Lemma 21 with "r" and "i" inserted in appropriate places.

4 PCF extended with real numbers

We introduce two successive extensions of PCF, first with a type for the lazy unit interval (Subsection 4.2) and then with a further type for the lazy real line (Subsection 4.3). We also discuss the issues of canonical evaluation and of deterministic evaluation of PCF programs containing parallel features (Subsections 4.4 and 4.5 respectively).

For the reader's convenience we introduce the basic notions of PCF needed in this paper (Subsection 4.1). See [19] for an excellent textbook account to PCF.

4.1 The programming language PCF

This section is based on [28], with minor adaptations and simplifications convenient to our needs, and can be safely skipped and used as a reference.

Given a collection of symbols called *ground types*, the set of *types* is the least set containing the ground types and containing the formal expression $(\sigma \to \tau)$ whenever it contains σ and τ. The greek letters σ and τ range over types. We let $(\sigma_1, \dots, \sigma_n, \tau)$ stand for $(\sigma_1 \to (\sigma_2 \to \cdots (\sigma_n \to \tau) \cdots))$.

Given a collection $\mathcal{L}$ of formal constants, each having a fixed type, and a family of formal variables $\{\alpha_i^\sigma\}$ $(i \geq 0)$ for each type σ, the $\mathcal{L}$*-terms* are given by the following inductive rules:

(i) Every variable α_i^σ is an $\mathcal{L}$-term of type σ.
(ii) Every constant of type σ is an $\mathcal{L}$-term of type σ.
(iii) 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.
(iv) 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 $\mathcal{L}$ is understood from the context it need not be used as a prefix. The letters L, M, and N range over terms. The letter c range over constants. We denote the fact that a term M has type σ by $M : \sigma$. Notice that every term has a unique type. Terms of the form (MN) are called *combinations*. Terms of the form $(\lambda\alpha M)$ are called *abstractions*. Parentheses around combinations and abstractions are sometimes omitted with the convention that juxtaposition associates to the left. We also omit parentheses in type expressions with the convention that $\to$ associates to the right.

The set of *free variables* of a term M is $\mathrm{FV}(M)$, inductively defined by

(i) $\mathrm{FV}(\alpha_i^\sigma) = \{\alpha_i^\sigma\}$
(ii) $\mathrm{FV}(c) = \emptyset$
(iii) $\mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$
(iv) $\mathrm{FV}(\lambda\alpha_i^\sigma M) = \mathrm{FV}(M) - \{\alpha_i^\sigma\}$

A term M is *closed* if $\mathrm{FV}(M) = \emptyset$ and *open* otherwise. *Programs* are closed terms of ground type. The idea is that the ground types are the data types and programs produce data, via the operational semantics. The remaining terms are significant as subprograms.

$[N/\alpha]M$ is the result of replacing all free occurrences of the variable α in M by N, making the appropriate changes in the bound variables of M so that no free variables of N become bound.

4.1.1 The languages $\mathcal{L}_{\mathrm{DA}}$, $\mathcal{L}_{\mathrm{PA}}$ and $\mathcal{L}_{\mathrm{PA}+\exists}$

The ground types for the language $\mathcal{L}_{\mathrm{DA}}$ are $\mathbf{N}$ and $\mathbf{T}$, and its constants are

$\mathbf{tt} : \mathbf{T} \quad \mathbf{ff} : \mathbf{T}$
$\mathbf{if}_\sigma : (\mathbf{T}, \sigma, \sigma, \sigma)$ for σ ground
$\mathbf{Y}_\sigma : (\sigma \to \sigma) \to \sigma$ for each σ
$\mathbf{k}_n : \mathbf{N}$ for each natural number n
$(+\mathbf{1}) : \mathbf{N} \to \mathbf{N}, \quad (-\mathbf{1}) : \mathbf{N} \to \mathbf{N}, \quad (= \mathbf{0}) : \mathbf{N} \to \mathbf{T}$

The language $\mathcal{L}_{\mathrm{PA}}$ is the language $\mathcal{L}_{\mathrm{DA}}$ extended by the constants

$\mathbf{pif}_\sigma : (\mathbf{T}, \sigma, \sigma, \sigma)$ for σ ground

The language $\mathcal{L}_{\mathrm{PA}+\exists}$ is the language $\mathcal{L}_{\mathrm{PA}}$ extended with the constant

$\exists : (\mathbf{N} \to \mathbf{T}) \to \mathbf{T}$

4.1.2 Denotational semantics

A *collection of domains for PCF* is a family $\{D_\sigma\}_\sigma$ of domains, one for each type σ, such that $D_{\sigma\to\tau} = [D_\sigma \to D_\tau]$. It is *standard* if $D_{\mathbf{T}}$ is the flat domain of truth values and $D_{\mathbf{N}}$ is the flat domain of natural numbers.

An *interpretation* of a language $\mathcal{L}$ is a collection $\{D_\sigma\}_\sigma$ of domains for PCF together with a mapping

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

which is type-respecting, in the sense that if $c : \sigma$ then $\mathcal{A}[\![c]\!] \in D_\sigma$.

An interpretation is *standard* if it interprets the constants $\mathbf{tt}$, $\mathbf{ff}$, $\mathbf{if}_\sigma$, $\mathbf{Y}_\sigma$, $\mathbf{k}_n$, $(+\mathbf{1})$, $(-\mathbf{1})$, $(= \mathbf{0})$, $\mathbf{pif}_\sigma$ and $\exists$ respectively as tt, ff, the sequential conditional, the fixed point operator, the natural number n, the successor function, the predecessor function, the test for zero, the parallel conditional and the continuous existential quantifier (whose definition is deliberately omitted).

An interpretation $\langle \{D_\sigma\}_\sigma, \mathcal{A} \rangle$ induces a *denotational semantics* $\hat{\mathcal{A}}$ for $\mathcal{L}$.

First, the set Env of *environments* is the set of type-respecting functions from the set of variables to $\bigcup_\sigma \{D_\sigma\}$. It is ranged over by ρ. If $\alpha : \sigma$ and $x \in D_\sigma$ then $\rho[x/\alpha]$ is the environment which maps α to x and any other variable α' to $\rho(\alpha')$. The *undefined environment* $\perp$ maps each variable of type σ to the bottom element of the domain D_σ.

The denotational semantics

$$M \mapsto \big(\rho \mapsto \hat{\mathcal{A}}[\![M]\!](\rho)\big) : \text{Terms} \to (\text{Env} \to \bigcup_\sigma \{D_\sigma\})$$

is inductively defined by:

(i) $\hat{\mathcal{A}}[\![\alpha]\!](\rho) = \rho(\alpha)$
(ii) $\hat{\mathcal{A}}[\![c]\!](\rho) = \mathcal{A}[\![c]\!]$
(iii) $\hat{\mathcal{A}}[\![MN]\!](\rho) = \hat{\mathcal{A}}[\![M]\!](\rho)\left(\hat{\mathcal{A}}[\![N]\!](\rho)\right)$
(iv) $\hat{\mathcal{A}}[\![\lambda\alpha M]\!](\rho)(x) = \hat{\mathcal{A}}[\![M]\!](\rho[x/\alpha])$ (with $x \in D_\sigma$ if $\alpha : \sigma$)

Informally,

(i) A variable denotes what the environment assigns to it.
(ii) A constant denotes what the interpretation assigns to it.
(iii) If a term M denotes the function $f : D \to E$ and a term N denotes the value $x \in D$ then the combination MN denotes the value $f(x) \in E$.
(iv) If a term M denotes the value $y_x \in E$ in an environment which assigns the value $x \in D$ to the variable α, then the abstraction $\lambda\alpha M$ denotes the function $f : D \to E$ defined by $f(x) = y_x$.

If M is closed then its denotation does not depend on the environment, in the sense that $\hat{\mathcal{A}}[\![M]\!](\rho) = \hat{\mathcal{A}}[\![M]\!](\rho')$ for all ρ and ρ'.

In order to simplify notation, we let $[\![M]\!]$ stand for the denotation $\hat{\mathcal{A}}[\![M]\!](\perp)$ of a closed term M with respect to an implicit semantics $\hat{\mathcal{A}}$. Also, for any term M, we let $[\![M]\!](\rho)$ stand for $\hat{\mathcal{A}}[\![M]\!](\rho)$.

4.1.3 Operational semantics

The operational semantics of $\mathcal{L}_{\text{DA}}$ is given by an *immediate reduction relation*, defined by the following rules:

(i) $(\lambda\alpha M)N \to [N/\alpha]M$
(ii) $\mathbf{Y}M \to M(\mathbf{Y}M)$
(iii) $(+\mathbf{1})\mathbf{k}_n \to \mathbf{k}_{n+1}, \quad (-\mathbf{1})\mathbf{k}_{n+1} \to \mathbf{k}_n$
(iv) $(=\mathbf{0})\mathbf{k}_0 \to \mathbf{tt}, \quad (=\mathbf{0})\mathbf{k}_{n+1} \to \mathbf{ff}$
(v) $\mathbf{if}\,\mathbf{tt}MN \to M, \quad \mathbf{if}\,\mathbf{ff}MN \to N$

(vi) $\dfrac{M \to M'}{MN \to M'N}, \quad \dfrac{N \to N'}{MN \to MN'}$ if M is **if**, $(+\mathbf{1})$, $(-\mathbf{1})$ or $(=\mathbf{0})$

We omit the reduction rules of the languages $\mathcal{L}_{\text{PA}}$ and $\mathcal{L}_{\text{PA}+\exists}$. The reduction relation preserves types, in the sense that if $M \to M'$ and M has type σ, so does M'.

Evaluation is given by a partial function Eval from programs to constants, defined by

$$\mathrm{Eval}(M) = c \text{ iff } M \to^* c$$

It is well-defined because if $M \to^* c$ and $M \to^* c'$ then $c = c'$.

The following theorem is often referred to as the *Adequacy Property* of PCF. It asserts that the operational and denotational semantics coincide.

Theorem 27 (Plotkin [28], Theorem 3.1) *For any $\mathcal{L}_{\mathrm{DA}}$ program M and constant c,*

$$\mathrm{Eval}(M) = c \textit{ iff } [\![M]\!] = [\![c]\!].$$

Proof. Lemma 29 below. □

The following definitions are introduced to formulate and prove Lemma 29.

The predicates Comp_σ are defined by induction on types by:

(i) If $M : \sigma$ is a program then M has property Comp_σ iff $[\![M]\!] = [\![c]\!]$ implies $\mathrm{Eval}(M) = c$.

(ii) If $M : \sigma \to \tau$ is a closed term it has property $\mathrm{Comp}_{\sigma\to\tau}$ iff whenever $N : \sigma$ is a closed term with property Comp_σ then MN is a term with property Comp_τ.

(iii) If $M : \sigma$ is an open term with free variables $\alpha_1 : \sigma_1, \ldots, \alpha_n : \sigma_n$ then it has property Comp_σ iff $[N_1/\alpha_1] \cdots [N_2/\alpha_n]M$ has property Comp_σ whenever $N_1, \ldots, N_n$ are closed terms having properties Comp_{σ_1}, …, Comp_{σ_n} respectively.

A term of type σ is *computable* if it has property Comp_σ.

If $M : \sigma \to \tau$ and $N : \sigma$ are closed computable terms, so is MN and also a term $M : (\sigma_1, \ldots, \sigma_n, \tau)$ is computable iff $\tilde{M} N_1 \ldots N_n$ is computable whenever the terms $N_1 : \sigma_1, \ldots, N_n : \sigma_n$ are closed computable terms and $\tilde{M}$ is a closed instantiation of M by computable terms.

In the following definitions α has to be chosen as some variable of appropriate type in each instance. Define terms Ω_σ by $\Omega_\sigma = \mathbf{Y}_\sigma(\lambda\alpha\alpha)$ for σ ground and $\Omega_{\sigma\to\tau} = \lambda\alpha\Omega_\tau$, and define terms $\mathbf{Y}_\sigma^{(n)}$ by $\mathbf{Y}_\sigma^{(0)} = \Omega_\sigma$ and $\mathbf{Y}_\sigma^{(n+1)} = \lambda\alpha(\mathbf{Y}_\sigma^{(n)}\alpha)$. Then $[\![\mathbf{Y}_\sigma]\!] = \bigsqcup_n [\![\mathbf{Y}_\sigma^n]\!]$ for any standard interpretation.

Now define the *syntactic information order* $\preccurlyeq$ as the least relation between terms such that

(i) $\Omega_\sigma \preccurlyeq M : \sigma$ and $\mathbf{Y}_\sigma^{(n)} \preccurlyeq \mathbf{Y}_\sigma$,
(ii) $M \preccurlyeq M$, and
(iii) if $M \preccurlyeq M' : \sigma \to \tau$ and $N \preccurlyeq N' : \sigma$ then $\lambda\alpha N \preccurlyeq \lambda\alpha N'$ and also $MN \preccurlyeq M'N'$.

Lemma 28 (Plotkin [28], Lemma 3.2) *If $M \preccurlyeq N$ and $M \to M'$ then either $M' \preccurlyeq N$ or else for some N', $N \to N'$ and $M' \preccurlyeq N'$.*

Proof. By structural induction on M and cases according to why the immediate reduction $M \to M'$ takes place. $\square$

We include the proof of the following lemma since we are going to extend it to PCF with real numbers:

Lemma 29 (Plotkin [28], Lemma 3.3) *Every $\mathcal{L}_{\mathrm{DA}}$-term is computable.*

Proof. By structural induction on the formation rules of terms:

(1) Every variable is computable since any closed instantiation of it by a computable term is computable.

(2) Every constant other than the $\mathbf{Y}_\sigma$'s is computable. This is clear for constants of ground type. Out of $+\mathbf{1}$, $-\mathbf{1}$, $=\mathbf{0}$, and **if** we only consider $-\mathbf{1}$ as an example. It is enough to show $(-\mathbf{1})M$ computable when M is a closed computable term of type $\mathbf{N}$. Suppose $[\![(-\mathbf{1})M]\!] = [\![c]\!]$. Then $c = \mathbf{k}_m$ for some m and so $[\![M]\!] = m + 1$. Therefore as M is computable, $M \to^* \mathbf{k}_{m+1}$ and so $(-\mathbf{1})M \to^* \mathbf{k}_m = c$.

(3) If $M : \sigma \to \tau$ and $N : \sigma$ are computable, so is the combination MN. If MN is closed so are M and N and its computability follows from clause ii of the definition of computability. If it is open, any closed instantiation L of it by computable terms has the form $\tilde{M}\tilde{N}$ where $\tilde{M}$ and $\tilde{N}$ are closed instantiations of M and N respectively and therefore themselves computable which in turn implies the computability of L and hence of MN.

(4) If $M : \tau$ is computable, so is the abstraction $\lambda\alpha M$. It is enough to show that the ground term $LN_1 \cdots N_n$ is computable when $N_1, \cdots, N_n$ are closed computable terms and L is a closed instantiation of $\lambda\alpha M$ by computable terms. Here L must have the form $\lambda\alpha\tilde{M}$ where $\tilde{M}$ is an instantiation of all free variables of M, except α, by closed computable terms. If $[\![LN_1 \cdots N_n]\!] =$

$[\![c]\!]$, then we have $[\![[N_1/\alpha]\tilde{M}N_2 \cdots N_n]\!] = [\![LN_1 \cdots N_n]\!] = [\![c]\!]$. But $[N_1/\alpha]\tilde{M}$ is computable and so too therefore is $\tilde{M}N_2 \cdots N_n$. Therefore $LN_1 \cdots N_n \rightarrow [N_1/\alpha]\tilde{M}N_2 \cdots N_n \rightarrow^* c$, as required.

(5) Each $\mathbf{Y}_\sigma$ is computable. It is enough to prove $\mathbf{Y}_\sigma N_1 \cdots N_k$ is computable when $N_1, \cdots, N_k$ are closed computable terms and $\mathbf{Y}_\sigma N_1 \cdots N_k$ is ground. Suppose $[\![\mathbf{Y}N_1 \cdots N_k]\!] = [\![c]\!]$. Since $[\![\mathbf{Y}_\sigma]\!] = \bigsqcup_n [\![\mathbf{Y}_\sigma^n]\!]$, $[\![\mathbf{Y}^{(n)}N_1 \cdots N_k]\!] = [\![c]\!]$ for some n. Since $[\![\Omega_\sigma]\!] = \perp$ for σ ground, Ω_σ is computable for σ ground. From this and (1), (3), and (4) proved above it follows that every Ω_σ and $\mathbf{Y}_\sigma^{(n)}$ is computable. Therefore $\mathbf{Y}^{(n)}N_1 \cdots N_k \rightarrow^* c$ and so by Lemma 28, $\mathbf{Y}N_1 \cdots N_k \rightarrow^* c$, concluding the proof. $\square$

This lemma extends to the languages $\mathcal{L}_{\mathrm{PA}}$ and $\mathcal{L}_{\mathrm{PA}+\exists}$. It suffices to show that **pif** and $\exists$ are computable, with respect to appropriate reduction rules.

4.2 The programming language PCF^I

We let $\mathcal{L}$ range over the languages $\mathcal{L}_{\mathrm{DA}}$, $\mathcal{L}_{\mathrm{PA}}$ and $\mathcal{L}_{\mathrm{PA}+\exists}$, and $\mathcal{L}^I$ denote the extension of $\mathcal{L}$ with a new ground type $\mathbf{I}$ and the following new constants:

(i) $\mathbf{cons}_a : \mathbf{I} \rightarrow \mathbf{I}$
(ii) $\mathbf{tail}_a : \mathbf{I} \rightarrow \mathbf{I}$
(iii) $\mathbf{head}_r : \mathbf{I} \rightarrow \mathbf{T}$
(iv) $\mathbf{pif}_\mathbf{I} : (\mathbf{T}, \mathbf{I}, \mathbf{I}, \mathbf{I})$

for each non-bottom $a \in \mathcal{I}$ with distinct rational end-points and each rational $r \in (0, 1)$. We refer to the ground type $\mathbf{I}$ as the *real number type*, and to programs of real number type as *real programs.*

4.2.1 Denotational semantics

We let $D_\mathbf{I} = \mathcal{I}$ and we extend the standard interpretation $\mathcal{A}$ of $\mathcal{L}$ to $\mathcal{L}^I$ by

(i) $\mathcal{A}[\![\mathbf{cons}_a]\!] = \mathrm{cons}_a$
(ii) $\mathcal{A}[\![\mathbf{tail}_a]\!] = \mathrm{tail}_a$
(iii) $\mathcal{A}[\![\mathbf{head}_r]\!] = \mathrm{head}_r$
(iv) $\mathcal{A}[\![\mathbf{pif}_\mathbf{I}]\!]pxy = \text{pif } p \text{ then } x \text{ else } y$

4.2.2 Operational semantics

We extend the immediate reduction relation $\to$ of $\mathcal{L}$ to $\mathcal{L}^I$ by the following rules:

$$
\begin{array}{rl}
\text{(i)} & \mathbf{cons}_a(\mathbf{cons}_b M) \to \mathbf{cons}_{ab}\, M \\
\text{(ii)} & \mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{Y}\mathbf{cons}_L \quad \text{if } b \le a \\
\text{(iii)} & \mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{Y}\mathbf{cons}_R \quad \text{if } b \ge a \\
\text{(iv)} & \mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{cons}_{b/a}\, M \quad \text{if } a \sqsubseteq b \text{ and } a \ne b \\
\text{(v)} & \mathbf{tail}_a(\mathbf{cons}_b M) \to \mathbf{cons}_{(a \sqcup b)/a}(\mathbf{tail}_{(a\sqcup b)/b}\, M) \\
 & \quad \text{if } a \simeq b,\ a \not\sqsubseteq b,\ b \not\sqsubseteq a,\ b \not\le a, \text{ and } a \not\le b
\end{array}
$$

$$
\begin{array}{rl}
\text{(vi)} & \mathbf{head}_r(\mathbf{cons}_a M) \to \mathbf{tt} \quad \text{if } a < r \\
\text{(vii)} & \mathbf{head}_r(\mathbf{cons}_a M) \to \mathbf{ff} \quad \text{if } a > r \\
\text{(viii)} & \mathbf{pif}\,\mathbf{tt} MN \to M, \quad \mathbf{pif}\,\mathbf{ff} MN \to N \\
\text{(ix)} & \mathbf{pif}\ L\ (\mathbf{cons}_a M)\ (\mathbf{cons}_b N) \to \\
 & \qquad\qquad \mathbf{cons}_{a \sqcap b}(\mathbf{pif}\ L\ (\mathbf{cons}_{a/(a\sqcap b)}\, M)\ (\mathbf{cons}_{b/(a \sqcap b)}\, N)) \\
 & \quad \text{if } a \sqcap b \ne \bot
\end{array}
$$

$$
\text{(x)}\quad \frac{N \to N'}{MN \to MN'} \quad \text{if } M \text{ is } \mathbf{cons}_a,\ \mathbf{tail}_a,\ \mathbf{head}_r \text{ or } \mathbf{pif}
$$

$$
\text{(xi)}\quad \frac{M \to M'}{\mathbf{pif} LM \to \mathbf{pif} LM'} \qquad \frac{N \to N'}{\mathbf{pif} LMN \to \mathbf{pif} LMN'}
$$

These rules are well-defined by virtue of Lemma 3 and because the extra conditions on them which are not present in Lemma 21 ensure that no bottom elements and no maximal elements are produced as subscripts of **cons** or **tail**. It may seem that there is a missing self-evident reduction rule for $\mathbf{tail}_a$, namely the rule $\mathbf{tail}_a(\mathbf{cons}_a M) \to M$; see remark after Lemma 35.

Notice that the immediate reduction rules for the parallel conditional are non-deterministic. The following lemma shows that this non-determinism does not produce inconsistencies:

Lemma 30 *$M \to N$ implies $[\![M]\!](\rho) = [\![N]\!](\rho)$ for all terms M and N and any environment ρ.*

Proof. This is true for the language $\mathcal{L}$, and remains true for the extended language $\mathcal{L}^I$ by virtue of Lemma 21. □

The partial map Eval on programs M of truth-value and natural number type is defined in the same way as for $\mathcal{L}$. It is well-defined by virtue of Lemma 30, because no two different constants have the same interpretation. We extend

Eval to a multi-valued map on real programs M by

$$\mathrm{Eval}(M) = \{a \in \mathcal{I} | M \to^* \mathbf{cons}_a M' \text{ for some } M'\};$$

that is, $a \in \mathrm{Eval}(M)$ iff M has a head-normal form $\mathbf{cons}_a M'$.

4.2.3 Soundness of the operational semantics

Lemma 31 *For all real programs M, $a \in \mathrm{Eval}(M)$ implies $a \sqsubseteq [\![M]\!]$.*

Proof. By Lemma 30, if $M \to^* \mathbf{cons}_a M'$ then $[\![M]\!] = [\![\mathbf{cons}_a M']\!] = a[\![M']\!]$. Therefore $a \sqsubseteq [\![M]\!]$, because the information order on partial numbers coincides with the prefix preorder. $\square$

Theorem 32 (Soundness) *For any real program M,*

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

Proof. The join exists because $[\![M]\!]$ is an upper bound of the set $\mathrm{Eval}(M)$. Therefore the result follows from Lemma 31. $\square$

4.2.4 Completeness of the operational semantics

By virtue of Lemma 31, for any real program M we have that $\mathrm{Eval}(M)$ is empty if $[\![M]\!] = \bot$. The following theorem states a stronger form of the converse:

Theorem 33 (Completeness) *For any real program M,*

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

Proof. Lemma 35 below. $\square$

The soundness and completeness properties in the senses of Theorems 32 and 33 can be referred together as the *Adequacy Property.*

Theorem 34 *PCF extended with real numbers enjoys the Adequacy Property.*

The recursive definitions given in Examples 18, 19, 20, 24, and 25 give rise to PCF programs in the obvious way. It would be a formidable task to syntactically prove the correctness of the resulting programs by appealing to the operational semantics given by the reduction rules. However, a mathematical proof of the correctness of a recursive definition allows us to conclude that the induced programs indeed produce correct results, via an application of the Adequacy Theorem. In fact, this is the point of denotational semantics.

We extend the inductive definition of the predicates Comp_σ by the following clause:

> A real program M has property $\mathrm{Comp}_{\mathbf{I}}$ if for every non-bottom partial number $x \ll [\![M]\!]$ (as close to $[\![M]\!]$ as we please) there is some $a \in \mathrm{Eval}(M)$ with $x \sqsubseteq a$.

If we read the relation $x \ll y$ as "x is a piece of information about y" then the above definition says that a real program is computable if every piece of information about its value can be produced in a finite number of reduction steps.

The domain-theoretic justificative for continuity of computable functions is that it is a finiteness condition [29,32,34,35]; a function f is continuous if a finite amount of information about $f(x)$ depends only on a finite amount of information about x. The continuity of a domain, which amounts to its way-below relation being well-behaved, gives us a general and abstract framework for consistently talking about "pieces of information". Then it should come as no surprise that Lemma 35 makes essential use of the way-below relation and of the continuity of the primitive functions.

The following lemma, which extends Lemma 29, establishes the Completeness Theorem:

Lemma 35 *Every term is computable.*

Proof. It suffices to extend the inductive proof of Lemma 29. We have to:

(i) slightly modify the proof for the case of abstractions, because it mentions constants and we have added new constants;
(ii) extend the proof of computability of the terms $\mathbf{Y}_\sigma$ to the new types; and
(iii) show that the new constants are computable.

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

It is enough to show that the ground term $LN_1 \cdots N_n$ is computable when $N_1, \cdots, N_n$ are closed computable terms and L is a closed instantiation of $\lambda\alpha M$ by computable terms. Here L must have the form $\lambda\alpha\tilde{M}$ where $\tilde{M}$ is an instantiation of all free variables of M, except α, by closed computable terms. Since $[N_1/\alpha]\tilde{M}$ is computable, so is $[N_1/\alpha]\tilde{M}N_2 \cdots N_n$. Since $LN_1 \cdots N_n \to [N_1/\alpha]\tilde{M}N_2 \cdots N_n$ and the reduction relation preserves meaning, in order to evaluate $LN_1 \cdots N_n$ it suffices to evaluate $[N_1/\alpha]\tilde{M}N_2 \cdots N_n$.

(ii) $\mathbf{Y}_\sigma$ is computable for all new types:

In order to prove that $\mathbf{Y}_\sigma$ is computable for all new types it suffices to show that the term $\mathbf{Y}_{(\sigma_1,\ldots,\sigma_k,\mathbf{I})}N_1 \cdots N_k$ is computable whenever $N_1 : \sigma_1, \ldots, N_k : \sigma_k$ are closed computable terms.

It follows from (i) above that the terms $\mathbf{Y}^{(n)}_\sigma$ are computable for all new types, because the proof of computability of $\mathbf{Y}^{(n)}_\sigma$ for old types depends only on the fact that variables are computable and that the combination and abstraction formation rules preserve computability.

The syntactic information order $\preccurlyeq$ on $\mathcal{L}$ extends to $\mathcal{L}^I$, and Lemma 28 is easily seen to remain true by a routine extension of its inductive proof.

Let $x \ll [\![\mathbf{Y}N_1 \cdots N_k]\!]$ be a non-bottom partial number. By a basic property of the way-below relation of any *continuous* dcpo, there is some n such that $x \ll [\![\mathbf{Y}^{(n)}N_1 \cdots N_k]\!]$, because $[\![\mathbf{Y}]\!] = \bigsqcup_n [\![\mathbf{Y}^{(n)}]\!]$. Since $\mathbf{Y}^{(n)}$ is computable, there is a $c \in \mathrm{Eval}(\mathbf{Y}^{(n)}N_1 \cdots N_k)$ with $x \sqsubseteq c$. Since there is a term M with $\mathbf{Y}^{(n)}N_1 \cdots N_k \to^* \mathrm{cons}_c M$ and $\mathbf{Y}^{(n)} \preccurlyeq \mathbf{Y}$, it follows from Lemma 28 that $\mathbf{Y}N_1 \cdots N_k \to^* \mathrm{cons}_c M$ for some M and therefore $c \in \mathrm{Eval}(\mathbf{Y}N_1 \cdots N_k)$.

(iii) The new constants are computable:

In order to prove that one of the new constants c is computable it suffices to show that if $M_1, \ldots, M_n$ are closed computable terms such that $cM_1 \ldots M_n$ has real number type, then $cM_1 \ldots M_n$ is computable.

(iii)(a) $\mathbf{cons}_a$ is computable:

Let M be a computable real program and let $x \ll [\![\mathbf{cons}_a M]\!] = a[\![M]\!]$ be a non-bottom partial number. We have to produce $c \in \mathrm{Eval}(\mathbf{cons}_a M)$ with $x \sqsubseteq c$. Let $b \ll [\![M]\!]$ with $x \ll ab$. If $b = \bot$ then we can take $c = a$. Otherwise let $b' \in \mathrm{Eval}(M)$ with $b \sqsubseteq b'$. Then we can take $c = ab'$, because $\mathbf{cons}_a M \to^* \mathbf{cons}_a(\mathbf{cons}_{b'} M') \to \mathbf{cons}_{ab'} M'$ for some M'.

(iii)(b) $\mathbf{tail}_a$ is computable:

Let M be a computable real program and let $y \ll [\![\mathbf{tail}_a M]\!] = \mathrm{tail}_a([\![M]\!])$ be a non-bottom partial number. We have to produce $c \in \mathrm{Eval}(\mathbf{tail}_a M)$ with $y \sqsubseteq c$. Since tail_a is continuous, there is some $x \ll [\![M]\!]$ such that $y \ll \mathrm{tail}_a(x)$. Let $b \in \mathrm{Eval}(M)$ with $x \sqsubseteq b$. It follows that $b \not\sqsubseteq a$, because $y \neq \bot$ and if $b \sqsubseteq a$ then $y \ll \mathrm{tail}_a(x) \sqsubseteq \mathrm{tail}_a(b) \sqsubseteq \mathrm{tail}_a(a) = \bot$. Then exactly one of the following four cases holds:

(t1) $b \leq a$

(t2) $a \leq b$
(t3) $a \sqsubseteq b$
(t4) $a \simeq b$, $a \not\sqsubseteq b$, $b \not\leq a$, and $a \not\leq b$

which have the following proofs:

(t1) In this case $\mathbf{tail}_a M \to^* \mathbf{tail}_a(\mathbf{cons}_b M') \to \mathbf{Ycons}_L$ for some M'. Hence $y \ll [\![\mathbf{tail}_a M]\!] = [\![\mathbf{Ycons}_L]\!]$. Since $\mathbf{Ycons}_L$ is a computable term, there is some $c \in \mathrm{Eval}(\mathbf{Ycons}_L) \subseteq \mathrm{Eval}(\mathbf{tail}_a M)$ with $y \sqsubseteq c$.
(t2) This case is handled similarly.
(t3) Since $x \sqsubseteq b$, we have that $\mathrm{tail}_a(x) \sqsubseteq \mathrm{tail}_a(b) = b/a$, and since $y \sqsubseteq \mathrm{tail}_a(x)$, we have that $y \sqsubseteq b/a$. Therefore we can take $c = b/a$, because $\mathbf{tail}_a M \to^* \mathbf{tail}_a(\mathbf{cons}_b M') \to \mathbf{cons}_{b/a} M'$ for some M'.
(t4) Since $x \sqsubseteq b$, we have that $\mathrm{tail}_a(x) \sqsubseteq \mathrm{tail}_a(b) = (a \sqcup b)/a$, and since $y \sqsubseteq \mathrm{tail}_a(x)$, we have that $y \sqsubseteq (a \sqcup b)/a$. Therefore we can take $c = (a \sqcup b)/a$, because $\mathbf{tail}_a M \to^* \mathbf{tail}_a(\mathbf{cons}_b M')) \to \mathbf{cons}_{(a\sqcup b)/a}(\mathbf{tail}_{(a\sqcup b)/b} M')$ for some M'.

(iii)(c) $\mathbf{head}_r$ is computable:

Assume that $[\![\mathbf{head}_r M]\!] = \mathrm{head}_r([\![M]\!]) \neq \bot$ for a computable real program M. Then there is an $x \ll [\![M]\!]$ such that either $x < r$ or else $x > r$, because head_r is continuous. Let $c \in \mathrm{Eval}(M)$ with $x \sqsubseteq c$. Then either $c < r$ or else $c > r$. Hence there is some M' such that either $\mathbf{head}_r M \to^* \mathbf{head}_r(\mathbf{cons}_c M') \to \mathrm{tt}$ or else $\mathbf{head}_r M \to^* \mathbf{head}_r(\mathbf{cons}_c M') \to \mathbf{ff}$ respectively. Therefore if $x < r$ then $\mathrm{Eval}(\mathbf{head}_r M) = \mathbf{tt}$ and if $x > r$ then $\mathrm{Eval}(\mathbf{head}_r M) = \mathbf{ff}$.

(iii)(d) $\mathbf{pif}$ is computable:

It suffices to show that $\mathbf{pif} LMN$ is computable whenever $[\![L]\!] = \bot$ and M and N are computable programs, because the case $[\![L]\!] \neq \bot$ is immediate. Let $x \ll [\![\mathbf{pif} LMN]\!]$. Then $x \ll [\![M]\!] \sqcap [\![N]\!]$. Hence $x \ll [\![M]\!]$ and $x \ll [\![N]\!]$. Let $a \in \mathrm{Eval}(M)$ and $b \in \mathrm{Eval}(N)$ with $x \sqsubseteq a$ and $x \sqsubseteq b$. Then $x \sqsubseteq a \sqcap b$ and $a \sqcap b \in \mathrm{Eval}(\mathbf{pif} LMN)$, because $\mathbf{pif} LMN \to^* \mathbf{pif} L(\mathbf{cons}_a M)(\mathbf{cons}_b N) \to \mathbf{cons}_{a\sqcap b}(\mathbf{pif} L(\mathbf{cons}_{a/(a\sqcap b)} M)(\mathbf{cons}_{b/(a\sqcap b)} N))$.

This concludes the proof of Lemma 35. □

The proof for tail_a implicitly shows that a reduction rule $\mathbf{tail}_a(\mathbf{cons}_a M) \to M$ would be useless. This can be explicitly explained as follows. If M denotes $\bot$, so does $\mathbf{tail}_a(\mathbf{cons}_a M)$. Hence there is no point in reducing $\mathbf{tail}_a(\mathbf{cons}_a M)$ before reducing M to a head-normal form. If M has a head-normal form $\mathrm{cons}_c M'$, then $\mathbf{tail}_a(\mathbf{cons}_a M)$ reduces to $\mathbf{tail}_a(\mathbf{cons}_{ac} M')$, which in turn reduces to $\mathrm{cons}_c M'$. In practice, However, this rule (and possibly more rules) can be included for efficiency reasons.

4.3 The programming language PCF^R

We let $\mathcal{L}^R$ denote the extension of $\mathcal{L}^I$ with a new ground type $\mathbf{R}$ and the following new constants:

(i) $\mathbf{rrcons}_a : \mathbf{R} \to \mathbf{R}$
(ii) $\mathbf{ricons}_a : \mathbf{I} \to \mathbf{R}$
(iii) $\mathbf{irtail}_a : \mathbf{R} \to \mathbf{I}$
(iv) $\mathbf{rhead}_r : \mathbf{R} \to \mathbf{T}$
(v) $\mathbf{pif}_{\mathbf{R}} : (\mathbf{T}, \mathbf{R}, \mathbf{R}, \mathbf{R})$

for each $a \in \mathcal{R}$ with distinct rational end-points and each rational number r.

We let $D_{\mathbf{R}} = \mathcal{R}_\perp$ and we extend the standard interpretation $\mathcal{A}$ of $\mathcal{L}^I$ to $\mathcal{L}^R$ in the obvious way:

(i) $\mathcal{A}[\![\mathbf{rrcons}_a]\!] = \mathrm{rrcons}_a$
(ii) $\mathcal{A}[\![\mathbf{ricons}_a]\!] = \mathrm{ricons}_a$
(iii) $\mathcal{A}[\![\mathbf{irtail}_a]\!] = \mathrm{irtail}_a$
(iv) $\mathcal{A}[\![\mathbf{rhead}_r]\!]x = \mathrm{rhead}_r$
(v) $\mathcal{A}[\![\mathbf{pif}_{\mathbf{R}}]\!]pxy = \text{pif } p \text{ then } x \text{ else } y$

The reduction rules for $\mathcal{L}^R$ are given by Lemma 26, with the same restrictions as for the reduction rules for $\mathcal{L}^I$. The Computability Lemma 35 and the Adequacy Theorem 34 routinely generalize to $\mathcal{L}^R$.

4.4 Canonical evaluation

Although we have that for all real programs M and N,

$$[\![M]\!] = [\![N]\!] \text{ iff } \bigsqcup \mathrm{Eval}(M) = \bigsqcup \mathrm{Eval}(N),$$

it is *not* the case that $[\![M]\!] = [\![N]\!]$ iff $\mathrm{Eval}(M) = \mathrm{Eval}(N)$. (But $[\![M]\!] = [\![N]\!]$ iff $\mathrm{Eval}(M)$ and $\mathrm{Eval}(N)$ are cofinal.) Moreover, $\mathrm{Eval}(M)$ may contain some intentional information. For example, suppose that a program $M : \mathbf{I}$ denotes $1/2$. Then it can be the case that $\mathrm{Eval}(M)$ contains L (take $M = \mathbf{cons}_L(\mathbf{Y}\mathbf{cons}_R)$). In this case we can *observe* that $[\![M]\!] \le 1/2$, which is certainly not an *observable property* of $1/2$ [1,34,35,38]. In fact, this property is not observable in the evaluation of *all* programs denoting $1/2$ (the program $M = \mathbf{Y}\mathbf{cons}_C$ is a counter-example).

Both problems can be solved by hiding the intensional information contained in $\mathrm{Eval}(M)$ as follows. Add an immediate reduction rule

$$\mathbf{cons}_a M \to \mathbf{cons}_b(\mathbf{cons}_{a/b} M) \text{ if } b \sqsubseteq a \text{ and } b \neq a,$$

and define a *strong head-normal form* to be a real program of the form $\mathbf{cons}_a(\mathbf{cons}_b M)$ with $0 < b < 1$. It follows that a real program M has a strong head-normal form $\mathbf{cons}_a(\mathbf{cons}_b M')$ iff $a \ll [\![M]\!]$, because $a \ll x$ iff there is some b such that $0 < b < 1$ and $ab \sqsubseteq x$. Now, for all real programs M, define

$$\mathrm{Eval}'(M) = \{a \in \mathcal{I} | M \text{ has a strong head-normal form } \mathbf{cons}_a(\mathbf{cons}_b M')\}.$$

With this definition it is immediate that $\bigsqcup \mathrm{Eval}'(M) = \bigsqcup \mathrm{Eval}(M)$ and

$$[\![M]\!] = [\![N]\!] \text{ iff } \mathrm{Eval}'(M) = \mathrm{Eval}'(N),$$

because

$$\mathrm{Eval}'(M) = \{a \in \mathcal{I} | a \ll [\![M]\!] \text{ and } a \text{ has distinct rational end-points}\}.$$

It is also possible to show that, for any standard enumeration of the rational numbers, the set $\mathrm{Eval}'(M)$ is recursively enumerable. Recall that an enumeration $r_0, r_1, \ldots, r_n, \ldots$ of the rational numbers is *standard* if there are recursive maps $s, p, q : \mathbb{N} \to \mathbb{N}$ such that

$$r_n = (-1)^{s(n)} \, \frac{p(n)}{q(n)},$$

and recall that the basic operations and predicates on rational numbers are recursive with respect to any standard enumeration. In order to prove that $\mathrm{Eval}'(M)$ is recursively enumerable it is necessary to assign Gödel numbers to terms and show that the reflexive and transitive closure of the immediate reduction relation is recursively enumerable with respect to the Gödel numbering.

In practice we do not need this modification, because the intensional information present in $\mathrm{Eval}(M)$ is not accessible within the language. In fact, it is even desirable to shorten the set $\mathrm{Eval}(M)$; see the subsection below.

4.5 Deterministic evaluation

By imposing a strategy on the application of the reduction rules, it is possible to eliminate the non-determinism and turn $\mathrm{Eval}(M)$ into a finite or infinite sequence with the partial number $[\![M]\!]$ as its iterated concatenation, for any real program M. We omit the details; but see below. For efficiency reasons, any practical functional programming language for exact real number computation based on the above ideas should have its evaluation implemented in this way.

The papers [8] and [22] give practical and theoretical explanations for the need of non-deterministic or parallel features in real number computation when we do not allow intensional operations. Notice that Bishop and Bridges [6] need an intensional operation in order to do constructive analysis, even for defining the basic operations on real numbers.

The above strategy eliminates the non-determinism of evaluation at the cost of simulating the parallel evaluation of the condition and the branches of the parallel conditional, until either the condition becomes true or false, or else both branches become a head-normal form. Perhaps it is possible to make use of abstract interpretation [2] in order to eliminate some instances of the parallelism.

5 Concluding remarks and further work

A useful extension of PCF with real numbers should be fully abstract and have all computable elements and functions definable. We are currently investigating the definability credentials of our extension. Our notion of computability is taken from [33]. We already know that the computable real numbers, the basic arithmetic operations, some trigonometric functions, and the logarithmic functions are definable. We conjecture that $\mathcal{L}^{R}_{PA+\exists}$ has all computable elements and functions definable (and therefore it is fully abstract).

Such an extension would be useful to mechanize the ideas presented in [10–12]. In an ongoing joint work with A. Edalat [13], we have shown that it is relatively easy to extend the language presented in this paper with a primitive for Riemann integration, by using the methods presented in [11]. We are also investigating methods for handling the more general theory of integration presented in *loc. cit.* within the language introduced in this paper.

Although such an approach to exact real number computation may seem to produce inefficient programs, it nevertheless provides a theoretical tool to prove computability without resorting to awkward indexings or representations. Recall that we do not need to know the operational semantics of the language in order to program in it, although it may be desirable to know it in order to obtain efficient programs. Once we know how to solve a problem theoretically, we can proceed to translate the solution to a real-life programming *imperative* language. Moreover, we do think that it is possible to extend real-life *functional* programming languages such as Haskell, Miranda, ML [5,27] with a relatively efficient treatment of real numbers based on continuous words, so that the theoretical solution can be readily used in practice. In such an approach it is desirable to include the basic arithmetic operations and other usual functions occurring in analysis as primitive.

In an even more practical approach, we can restrict ourselves to the partial real numbers with floating-point end-points. But we cannot hope Theorem 33 to hold, due to the presence of round-off errors (equivalently, due to the fact that there are finitely many floating-point numbers). Nevertheless, any produced result is guaranteed to be a correct partial realization of the ideal result. We emphasize that the best produced result can be worse than the best machine representable partial realization, this time due to round-off errors alone. Moreover, two distinct programs with the same value can concretely evaluate to distinct partial realizations of their ideal value, again due to round-off errors. In fact, the (correct) restriction of the concatenation operation to floating-point partial numbers does not satisfy the associativity property; it only satisfies the subassociativity law $(xy)z \sqsubseteq x(yz)$, in such a way that the fundamental computation rule $\mathrm{cons}_a(\mathrm{cons}_b(x)) \to \mathrm{cons}_{ab}(x)$ introduces information loss. Such an implementation of PCF extended with real numbers is close to the ideas presented in [25]. Notice that Interval Analysis as introduced in *loc. cit.* remains an active area of research.

Since Automata Theory is based on monoids of words, it seems plausible that the monoid of continuous words could also be used to generalize automata to real numbers in a natural way.

Acknowledgement

This work has been carried out under the supervision of Mike Smyth. I have benefited from endless discussions with him and Abbas Edalat. Several useful comments by Abbas Edalat on the draft versions of this paper have allowed me to improve the presentation. The author has been supported by the Brazilian agency CNPq.

References

[1] S. Abramsky. *Domain Theory and the Logic of Observable Properties.* PhD thesis, University of London, Queen's College, 1987.

[2] S. Abramsky. Abtract interpretation, logical relations and Kan extensions. *Journal of Logic and Computation*, 1(1):5–40, 1990.

[3] 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, Oxford, 1994.

[4] B.M. Acióly. *A Computational Foundation of Interval Mathematics.* PhD thesis, Universidade Federal do Rio Grande do Sul, Instituto de Informatica, Porto Alegre, Brazil, 1991. (In Portuguese).

[5] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice-Hall, New York, 1988.

[6] E. Bishop and D. Bridges. *Constructive Analysis.* Springer-Verlag, Berlin, 1985.

[7] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers. *Bull. Amer. Math. Soc.*, 21:1–46, 1989.

[8] H.J. Boehm, R. Cartwright, M. Riggle, and M.J. O'Donnel. Exact real arithmetic: A case study in higher order programming. In *ACM Symposium on Lisp and Functional Programming*, 1986.

[9] P. di Gianantonio. *A Functional Approach to Computability on Real Numbers.* PhD thesis, Università Degli Studi di Pisa, Dipartamento di Informatica, 1993.

[10] A. Edalat. Power domains and iterated function systems. Technical Report Doc 94/13, Department of Computing, Imperial College, 1994. Submitted to *Information and Computation.*

[11] A. Edalat. Domain theory and integration. *Theoretical Computer Science*, 151:163–193, 1995.

[12] A. Edalat. Dynamical systems, measures and fractals via domain theory. *Information and Computation*, 120(1):32–48, July 1995.

[13] A. Edalat and M.H. Escardó. Real PCF extended with integration. Unpublished draft presented in the Computational Measure and Integration Theory Workshop, Imperial College, London (available by anonymous ftp at theory@doc.ic.ac.uk:papers/Escardo/drafts), October 1995.

[14] M.H. Escardó. Induction and recursion on the real line. In C. Hankin, I. Mackie, and R. Nagarajan, editors, *Theory and Formal Methods 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 259–282, Møller Centre, Cambridge, UK, 11–14 September 1994. IC Press. 1995.

[15] M.H. Escardó and D.M. Claudio. Scott domain theory as a foundation for interval analysis. Technical Report 218, UFRGS/II, Porto Alegre, Brazil, 1993.

[16] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, New York, 1980.

[17] K. Grue. Unrestricted lazy numerical algorithms. Unpublished manuscript.

[18] A. Grzegorczyk. On the definition of computable real continuous functions. *Fund. Math.*, 44:61–77, 1957.

[19] C. A. Gunter. *Semantics of Programming Languages – Structures and Techniques*. The MIT Press, London, 1992.

[20] J.G. Hocking and G.S. Young. *Topology*. Dover Publications, 1988.

[21] S.C. Kleene and R.E Vesley. *The Foundations of Intuitionistic Mathematics: Especially in Relation to Recursive Functions*. North-Holland, Amsterdam, 1965.

[22] H. Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, 5:321–324, 1977.

[23] S. MacLane and G. Birkhoff. *Algebra*. Macmillan, New York, 1st edition, 1967.

[24] P. Martin-Lof. *Notes on Constructive Mathematics*. Almqvist & Wiksell, Stockholm, 1970.

[25] R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, 1966.

[26] J. Myhill. Criteria of constructivity of real numbers. *J. Symbolic Logic*, 18:7–10, 1953.

[27] L.C. Paulson. *ML for the working programmer*. Cambridge University Press, Cambridge, 1991.

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

[29] G. Plotkin. Domains. Post-graduate Lectures in advanced domain theory, University of Edinburgh, Department of Computer Science, 1980.

[30] M.B. Pour-el and I. Richards. Computability and non-computability in classical analysis. *Trans. Am. Math. Soc.*, pages 539–560, 1983.

[31] H.G. Rice. Recursive real numbers. *Proc. Amer. Math. Soc.*, pages 784–791, 1954.

[32] D. S. Scott. Lattice theory, data types and semantics. In *Formal semantics of programming languages*, pages 66–106, Englewood Cliffs, 1972. Prentice-Hall.

[33] M.B. Smyth. Effectively given domains. *Theoretical Computer Science*, 5(1):256–274, 1977.

[34] M.B. Smyth. Power domains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming*, pages 662–675. Springer-Verlag, 1983. LNCS 154.

[35] M.B. Smyth. Topology. In S. Abramsky, D. M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 641–761. Clarendon Press, Oxford, 1992.

[36] M.B. Smyth. Semi-metrics, closure spaces, and digital topology. *Theoretical Computer Science*, 151:257–276, 1995.

[37] A. Turing. On computable numbers, with an application to the Entscheindungproblem. *The London Mathematical Society*, 42:230–265, 1936.

[38] S. Vickers. *Topology via Logic.* Cambridge University Press, Cambridge, 1989.

[39] J. Vuillemin. Exact real arithmetic with continued fractions. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 14–27, 1988.

[40] Klaus Weihrauch. *Computability.* Springer-Verlag, Berlin, 1987.

[41] Klaus Weihrauch. A simple introduction to computable analysis. Technical Report 171 – 7/1995, FernUniversitat, 1995.

[42] E. Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.

Contents