Induction and recursion on the partial real line with applications to Real PCF

Martín Hötzel Escardó

Department of computer science, Edinburgh University, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK, e-mail: mhedcs.ed.ac.uk.

Thomas Streicher

Fachbereich Mathematik, Technische Hochschule Darmstadt, Schloßgartenstraße 7, 64289 Darmstadt, Germany, e-mail: streichermathematik.th-darmstadt.de.

Abstract

The partial real line is an extension of the Euclidean real line with *partial* real numbers, which has been used to model exact real number computation in the programming language Real PCF. We introduce induction principles and recursion schemes for the partial unit interval, which allow us to verify that Real PCF programs meet their specification. They resemble the so-called Peano axioms for natural numbers. The theory is based on a domain-equation-like presentation of the partial unit interval. The principles are applied to show that Real PCF is universal in the sense that all computable elements of its universe of discourse are definable. These elements include higher-order functions such as integration operators.

Keywords: Induction, coinduction, exact real number computation, domain theory, Real PCF, universality.

Introduction

The partial real line is the domain of compact real intervals ordered by reverse inclusion [28,21]. The idea is that singleton intervals represent *total* real numbers, and that the remaining intervals represent (properly) *partial* real numbers. This is justified by the fact that the singleton map $x \mapsto \{x\}$ is a topological embedding of the Euclidean real line into the partial real line endowed with its Scott topology. The partial real line has been used to model exact real number computation in the framework of the programming language Real PCF [9,10], including computation of integrals [4].

We introduce induction principles and recursion schemes for the partial unit interval (the domain of closed subintervals of the unit interval with end-points 0 and 1), which allow us to verify that Real PCF programs meet their specification.

The induction principles and recursion schemes discussed in this paper resemble the so-called Peano axioms for natural numbers. They abstractly characterize the partial unit interval up to isomorphism, without reference to real numbers or intervals. Essentially, we replace zero and the successor function by the affine maps $x \mapsto x/2$ and $x \mapsto (x+1)/2$, which play the rôle of partial real number constructors. This is related to binary expansions and, perhaps surprisingly, to Dedekind sections at the same time.

Preliminary ideas on recursion and induction on the real line appeared in [8], which considers uniform spaces. Our axioms are formulated in the framework of domain theory [1].

Domain theory allows one to derive induction principles and recursion schemes from canonical solutions of domain equations [20,34,24]. Domain equations model recursive definitions of data types, such as lists and trees. Since the partial real line is not an algebraic domain, it is not the canonical solution of any domain equation involving usual functors.

We establish new results about the notion of an inductive retraction introduced in [10], which generalizes canonical solutions of domain equations by means of ideas similar to those of Freyd [14,15]. In particular, we introduce the notion of a biquotient of a bifree algebra, and we show that the inductive retractions are the biquotients of the bifree algebras.

An interesting observation is that the Peano-like axioms discussed above consider only induction and recursion, but the inductive retraction induced by them automatically gives rise to coinduction and corecursion.

The techniques discussed here in a more general setting were also applied in conjunction with the technique introduced in [36] to show that Real PCF extended with a certain computable existential quantifier is universal [10], in the sense that all computable elements of its universe of discourse are definable. These elements include higher-order functions such as supremum and integration operators. Here we consider a further extension of the language with recursive types.

The universality result depends on a notion of computability. In domain theory this is achieved via effective presentations [6,30]. It is straightforward to show that there exists an effective presentation of the partial real line that makes the primitive operations of the language computable. For example, any standard enumeration of the rational basis gives such an effective presentation. But

one then wonders whether a cleverer choice of an effective presentation would change the induced set of computable elements and functions, and this is indeed the case in general, as Kanda and Park have shown [19]. We apply the presentation of the partial real line as an inductive retract to show that there is a unique effective presentation of the partial real line that makes the primitive (and hence all) operations computable, up to equivalence of effective presentations. We can thus speak unambiguously about computability on the partial real line.

However, one still wonders how this relates to the classical theory of computability over the (total) real line. It was conjectured in [11] that a real valued function of real variables is computable iff it has a computable extension to a partial real valued function of partial real variables. It has been shown in [5] that this is indeed the case.

Organization

This paper is the full version of the extended abstracts [10,12]. It is organized in the following sections: (1) Domain theory, (2) The partial real line, (3) Peano-like axioms for the partial unit interval, (4) Generalized domain equations, (5) A generalized domain equation for the partial unit interval, (6) Applications to the programming language Real PCF.

1 Domain theory

Our main reference to domain theory is [1]. For its connections with topology see [33,16] (the papers [27] and [31] contain interesting technical and intuitive computational interpretations of topological concepts). Here we establish terminology and recall basic facts. Readers who are familiar with domain theory can proceed directly to Section 2.

1.1 Domains

A *poset* (partially ordered set) is a set equipped with a *partial order* (a reflexive, transitive and antisymmetric relation), generally denoted by the symbol $\sqsubseteq$. Existing joins (least upper bounds, suprema) and meets (greatest lower bounds, infima) are denoted by the symbols $\bigsqcup$ and $\bigsqcap$ respectively.

A subset Δ of a poset is said to be *directed* if each finite subset of Δ has an upper bound in Δ. Since the empty set is included in this definition, a directed

set is necessarily non-empty.

A *dcpo* (directed complete poset) is a poset with least upper bounds of directed subsets. A subset U of a dcpo D is *Scott open* it is an upper set and the condition $\bigsqcup \Delta \in U$ for $\Delta \subseteq D$ directed implies $\Delta \cap U \neq \emptyset$. The Scott open sets form a topology, known as the *Scott topology*. A function $f : D \to E$ between dcpos D and E is Scott continuous (continuous with respect to the Scott topologies) iff it is monotone and preserves least upper bounds of directed subsets.

A poset is *bounded complete* if it has a least element $\bot$ (bottom) and joins of bounded subsets. In the presence of directed completeness, bounded completeness is equivalent to the existence of all non-empty meets.

In order to ensure that some facts stated below are true, we need to assume that our dcpos are *continuous*. Moreover, continuity of dcpos is a fundamental ingredient of the formulation of the notion of an effectively given domain. However, since we use continuity of dcpos only implicitly, via the facts that it entails, we deliberately omit the elaborate usual definition in terms of the so-called way-below relation.

For the bounded complete case, which is what we need in this paper, a telegraphic characterization is available: A bounded complete dcpo D is continuous iff every $x \in D$ can be expressed as $\bigsqcup\{\bigsqcap U | x \in U\}$, where U ranges over Scott open subsets of D—see [27,16].

A familiar example from real analysis illustrates this characterization. The extended real line under its natural order is a bounded complete dcpo (in fact, a complete lattice). In this case the usual notation for suprema and infima is sup and inf. As one easily checks [16, pages 49–50], the Scott topology of this example is nothing but the topology of lower semicontinuity [26, pages 38, 50–52], given by non-trivial open sets of the form $(r, +\infty]$. Hence the above characterization reduces to the fact that every real number x can be expressed as $\sup\{\inf(r, \infty] | x \in (r, \infty]\} = \sup\{r | r < x\}$. Therefore the extended real line is a continuous lattice. In fact, this is a fundamental example. Continuity of the interval domain introduced below follows from the the same consideration on end-points of intervals. However, one has to admit that the example of the extended real line is a bit misleading in connection with the terminology *information order* introduced below. A better example is the interval domain.

In this paper a *domain* is a bounded complete continuous dcpo D. The order of a domain is referred to as its *information order*, and the relation $x \sqsubseteq y$ is interpreted as expressing the fact that y contains the same amount of information as x, and possibly more. Alternatively, one says that x is less defined than y. Under these interpretations, the bottom element has an empty information content, or is an undefined element. Computationally, bottom

denotes non-terminating computations which don't produce results.

A domain will be always considered as a topological space under its Scott topology. We often (implicitly) use the fact that the binary meet operation $\sqcap : D \times D \to D$ is (jointly) continuous for every domain D.

Trivial but useful examples of domains arise as follows. One starts with a set A, adds a new element $\bot$, and declares that $x \sqsubseteq y$ iff $x = \bot$ or $x = y$. The resulting domain is called a *flat domain* and is denoted by $A_\bot$. Typical examples are the flat domains of natural numbers and truth-values, $\mathcal{N} = \mathbb{N}_\bot$ and $\mathcal{B} = \mathbb{B}_\bot$, where $\mathbb{B} = \{\mathtt{true}, \mathtt{false}\}$.

1.2 Retracts and idempotents

A *retraction* is a continuous function $r : D \to E$ for which there exists a continuous map $s : E \to D$ with $r \circ s = \mathrm{id}_E$, called a *section*. In this case E is said to be a *retract* of D. Retractions are quotient maps and sections are subspace embeddings.

The map $s \circ r : D \to D$ is an idempotent whose fixed-points form a domain isomorphic to E. Conversely, every idempotent $e : D \to D$ gives rise to a domain $E = e(D)$ which is a retract of D with retraction given by the corestriction of e to E and section given by the restriction of e to E.

1.3 Recursive definitions of functions

Any continuous endofunction $f : D \to D$ of a dcpo D with a bottom element has a least fixed point given by $\mathrm{fix}(f) = \bigsqcup_{n \in \mathbb{N}} f^n(\bot)$.

If D and E are domains then the continuous functions from D to E, pointwise ordered by

$$f \sqsubseteq g \text{ iff } f(x) \sqsubseteq g(x) \text{ for all } x \in D,$$

form a domain denoted by $[D \to E]$. Existing joins and binary meets are computed pointwise.

Fixed points and function spaces are applied to solve functional equations of the form

$$f = F(f)$$

with $F : [D \to E] \to [D \to E]$ continuous. Such a functional equation is often

referred to as a *recursive definition* of $f : D \to E$, and it is implicit that the least solution is taken:

$$f = \mathrm{fix}(F).$$

1.4 Recursive definitions of domains

(The following material is not needed until Section 4.)

Domain theory also allows one to define *domains* by recursion:

$$D \cong \mathbf{F}D,$$

where $\mathbf{F}$ is an endofunctor of the category of domains. Domain equations are used to model recursively defined data types, such a lists and trees.

A solution of a domain equation $D \cong \mathbf{F}D$ is a domain D together with an isomorphism $i : \mathbf{F}D \to D$. In order to state what the *canonical* solution is, it is convenient to use some basic category theory [25].

Let $\mathbf{X}$ be a category (for example the category of domains and continuous functions) and $\mathbf{F} : \mathbf{X} \to \mathbf{X}$ be a functor.

An $\mathbf{F}$-*algebra* is an arrow $\kappa : \mathbf{F}X \to X$, and a *homomorphism* from an algebra $\kappa : \mathbf{F}X \to X$ to an algebra $a : \mathbf{F}A \to A$ is a map $h : X \to A$ with $h \circ \kappa = a \circ \mathbf{F}h$. Dually, an $\mathbf{F}$-*coalgebra* is an arrow $\delta : X \to \mathbf{F}X$, and a *homomorphism* from a coalgebra $b : B \to \mathbf{F}B$ to a coalgebra $\delta : X \to \mathbf{F}X$ is a map $k : B \to X$ with $\mathbf{F}k \circ b = \delta \circ k$. These definitions are illustrated in the following diagrams:

$$\begin{array}{ccc} \mathbf{F}X & \xrightarrow{\kappa} & X \\ {\scriptstyle \mathbf{F}h}\downarrow & & \downarrow{\scriptstyle h} \\ \mathbf{F}A & \xrightarrow[a]{} & A \end{array} \qquad \begin{array}{ccc} B & \xrightarrow{b} & \mathbf{F}B \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle \mathbf{F}k} \\ X & \xrightarrow[\delta]{} & \mathbf{F}X \end{array}$$

One writes $h : \kappa \to a$ and $k : b \to \delta$ to indicate that h and k are (co)algebra homomorphisms. Algebra (respectively coalgebra) homomorphisms compose and form a category.

If $i : \mathbf{F}X \to X$ is an initial algebra (in the sense that there is a unique homomorphism from it to any other algebra) then i is an isomorphism in $\mathbf{X}$.

A *canonical solution* of an equation $X \cong \mathbf{F}X$ is an initial algebra $i : \mathbf{F}C \to C$ whose inverse $i^{-1} : C \to \mathbf{F}C$ is a final coalgebra. Such an algebra is called a *bifree algebra*. By initiality, bifree algebras are unique up to (unique) isomorphism.

We now turn back to domains. A function between domains is *strict* if it preserves bottom elements. The category of domains and strict continuous functions is denoted by **SDom**. A functor $\mathbf{F} : \mathbf{SDom} \to \mathbf{SDom}$ is said to be *locally continuous* if for all domains D and E, the map $f \mapsto \mathbf{F}f : [D \to E] \to [\mathbf{F}D \to \mathbf{F}E]$ is continuous.

Every locally continuous functor $\mathbf{F} : \mathbf{SDom} \to \mathbf{SDom}$ has a bifree algebra $i : \mathbf{F}C \to C$. The reader is referred to [34,1] for its construction. But only the following facts are needed in this paper.

If $a : \mathbf{F}A \to A$ is an algebra and $b : B \to \mathbf{F}B$ is a coalgebra, then the unique homomorphisms $h : i \to a$ and $k : b \to i^{-1}$ can be recursively defined by

$$h = a \circ \mathbf{F}h \circ i^{-1}, \qquad k = i \circ \mathbf{F}k \circ b.$$

The functions h and k are said to be defined by structural recursion and corecursion respectively.

Structural recursion generalizes primitive recursion on the natural numbers, whereas structural corecursion generalizes minimization [32].

1.5 Effectively given domains

The reader is referred to [30]. The references [6,24] consider only algebraic domains, which exclude the partial real line. On the other hand, [5] considers a weaker version of [30]. Its drawback, from the point of view of this paper, is that effectively given domains in the weak sense are not closed under the function space construction. However, the reference contains the basic ingredients of [30] in an accessible form.

Essentially, an effective presentation of a domain is an enumerated basis of the domain, subject to certain axioms. An element of such an effectively given domain is defined to be computable if it is the join of a recursively enumerable directed set of basis elements. Computable functions can be defined as computable elements of function spaces, although equivalent direct definitions are also available.

2 The partial real line

The set $\mathcal{R} = \mathbf{I}\mathbb{R}$ of compact real intervals ordered by reverse inclusion is referred to as the ***partial real line***. It fails to be a domain only because it lacks a bottom element (which can be artificially added when necessary). The

elements of $\mathcal{R}$ are referred to as ***partial real numbers***, and a real number r is notationally identified with the singleton interval $\{r\}$ and referred to as a ***total real number***. Total real numbers have maximal information content.

The end-points of an interval x are given by

$$\underline{x} = \inf x, \qquad \overline{x} = \sup x.$$

By definition, the information order of $\mathcal{R}$ is given by

$$x \sqsubseteq y \text{ iff } x \supseteq y \text{ iff } \underline{x} \leq \underline{y} \text{ and } \overline{y} \leq \overline{x}.$$

Existing joins are given by

$$\bigsqcup X = \bigcap X = [\sup_{x \in X} \underline{x}, \inf_{x \in X} \overline{x}].$$

Binary meets are given by

$$x \sqcap y = [\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})].$$

This coincides with $x \cup y$ only if x and y intersect.

2.1 Canonical extensions

The singleton map $j : \mathbb{R} \to \mathcal{R}$ defined by $j(x) = \{x\}$ embeds the real line as a subspace of the partial real line. This is an embedding onto the total reals.

If $g : \mathcal{R} \to \mathcal{R}$ is a function which maps total reals to total reals, then it restricts to a function $f : \mathbb{R} \to \mathbb{R}$ as in the diagram

$$\begin{array}{ccc} \mathbb{R} & \xrightarrow{f} & \mathbb{R} \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle j} \\ \mathcal{R} & \xrightarrow[g]{} & \mathcal{R} \end{array}$$

What is interesting is that if g is continuous with respect to the Scott topology in addition, then f is continuous with respect to the Euclidean topology. This is the case precisely because $j : \mathbb{R} \to \mathcal{R}$ is a subspace embedding.

What is more interesting and does not follow from simple topological considerations such as the above is that the converse is also true: Every continuous map $f : \mathbb{R} \to \mathbb{R}$ extends to a continuous map $g : \mathcal{R} \to \mathcal{R}$ as in the above diagram.

(If a bottom element is added to $\mathcal{R}$, the extension property is an immediate consequence of the fact that the domains are characterized as the densely injective spaces [16, page 127] and that embeddings onto maximal elements are dense. In any case, $\mathcal{R}$ belongs to a larger class of domains, which are characterized precisely as the injective spaces over finitary subspace embeddings, which include $j : \mathbb{R} \to \mathcal{R}$ [7]. But in this paper we prefer to give a proof of the above special case of the extension property from first principles.)

Among all extensions there is a ***canonical extension*** $\mathbf{I}f : \mathcal{R} \to \mathcal{R}$ given by

$$\mathbf{I}f(x) = \{f(r) | r \in x\},$$

which is characterized as the most defined continuous extension, in the sense of the order of the function space $[\mathcal{R} \to \mathcal{R}]$. It is well-defined because continuous functions map connected sets to connected sets and compact sets to compact sets, and hence compact intervals to compact intervals. It is Scott continuous because the direct image-formation operation preserves filtered intersections of compact sets, which in this case correspond to directed joins.

If f is increasing with respect to the natural order of real numbers then its canonical extension is computed pointwise:

$$\mathbf{I}f(x) = [f(\underline{x}), f(\overline{x})].$$

A function $f : \mathbb{R} \to \mathbb{R}$ is notationally identified with its canonical extension $\mathbf{I}f : \mathcal{R} \to \mathcal{R}$, and we often define a continuous function $f : \mathcal{R} \to \mathcal{R}$ by first defining a continuous function $f : \mathbb{R} \to \mathbb{R}$ and then implicitly taking its canonical extension.

2.2 The partial unit interval

We shall work mainly with a subspace of the partial real line, the ***partial unit interval***, defined as the domain $\mathcal{I} = \mathbf{I}[0, 1]$ of closed subintervals of the unit interval $[0, 1]$ ordered by reverse inclusion. Its bottom element is $\bot = [0, 1]$. The singleton map embeds the unit interval as a subspace of the unit interval, and the extension property is preserved.

3 Peano-like axioms for the partial unit interval

In order to separate the main line of argumentation from the details of the individual lemmas, their proofs are collected together at the end of this section.

We define the *left* and *right successor* functions $\mathrm{succ}_L, \mathrm{succ}_R : \mathcal{I} \to \mathcal{I}$ by

$$\mathrm{succ}_L(x) = x/2, \qquad \mathrm{succ}_R(x) = (x+1)/2.$$

That is, succ_L and succ_R are the unique increasing affine maps which map the unit interval to its left and right halves $L = [0, 1/2]$ and $R = [1/2, 1]$:

$$\mathrm{succ}_L(\bot) = L, \qquad \mathrm{succ}_R(\bot) = R.$$

We regard succ_L and succ_R as partial real number "constructors" analogous to zero and the successor map $\mathrm{succ}(x) = x + 1$ on the natural numbers.

An important difference is that the natural numbers together with zero and successor form a free algebra, whereas the partial unit interval together with succ_L and succ_R will form a biquotient of a bifree algebra with respect to some equations. The main such equation is

$$\mathrm{succ}_L(1) = \mathrm{succ}_R(0),$$

which can be expressed as

$$\mathrm{succ}_L(\mathrm{fix}(\mathrm{succ}_R)) = \mathrm{succ}_R(\mathrm{fix}(\mathrm{succ}_L))$$

by observing that 0 and 1 are the unique fixed points of succ_L and succ_R.

Another important difference is that natural numbers are constructed from zero by *finitely* many applications of the successor function, whereas the elements of the partial unit interval are constructed by *infinitely* many applications of succ_L and succ_R. For example, every *total* $x \in \mathcal{I}$ can be constructed as

$$x = \bigsqcup_n \mathrm{succ}_{a_1} \circ \cdots \circ \mathrm{succ}_{a_n}(\bot)$$

for some sequence $a_i \in \{L, R\}$ corresponding to a binary expansion of x. Partial elements are constructed by iterating succ_L and succ_R in a more elaborate way, as it is shown in Section 5.3.

Pursuing our analogy with natural numbers, we now observe that every natural number is either zero or else the successor of a unique number.

For $x, y \in \mathcal{R}$, define

(i) $x < y$ iff $\overline{x} < \underline{y}$,
(ii) $x \leq y$ iff $\overline{x} \leq \underline{y}$,
(iii) $x \uparrow y$ iff x and y have an upper bound in the information order iff x and y intersect as intervals.

Then it is clear that *exactly one* of the following conditions holds:

$$x < y, \qquad x \uparrow y, \qquad x > y.$$

Lemma 1 (Dyadic Trichotomy) *For every $x \in \mathcal{I}$,*

(i) if $x \leq 1/2$ then $x = \mathrm{succ}_L(y)$ for a unique y,
(ii) if $x \geq 1/2$ then $x = \mathrm{succ}_R(z)$ for a unique z,
(iii) if $x \uparrow 1/2$ then $x = \mathrm{succ}_L(y) \sqcap \mathrm{succ}_R(z)$ for unique $y \sqsubseteq 1$ and $z \sqsubseteq 0$.

Recall that a number is notationally identified with a singleton interval, and notice that $x \uparrow 1/2$ iff $x \sqsubseteq 1/2$, because $1/2$ is maximal in information content.

The predecessor function $\mathrm{pred}(x) = x - 1$ on natural numbers, undefined or arbitrarily defined at zero, is a left inverse of the successor function. Similarly, succ_a has a left inverse pred_a defined by

$$\mathrm{pred}_L(x) = \min(2x, 1), \qquad \mathrm{pred}_R(x) = \max(0, 2x - 1).$$

The fact that every natural number is either zero or a successor can be expressed by the equation

$$n = \text{if } n = 0 \text{ then } 0 \text{ else } \mathrm{succ}(\mathrm{pred}(n)).$$

Let $\mathcal{B}$ be the flat domain of truth values, and for all $r \in [0, 1]$ define a continuous predicate $\mathrm{left}_r : \mathcal{I} \to \mathcal{B}$ by

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

where

$$(x <_\perp y) = \begin{cases} \texttt{true} & \text{if } x < y, \\ \texttt{false} & \text{if } x > y, \\ \perp & \text{if } x \uparrow y. \end{cases}$$

Also, write

$$\mathrm{left} = \mathrm{left}_{1/2}.$$

Remark 2 *In [9–12] the maps succ_a, pred_a and left_r are denoted by cons_a, tail_a and head_r respectively.* □

Finally, define the *parallel conditional* by

$$\text{pif } p \text{ then } x \text{ else } y = \begin{cases} x & \text{if } p = \texttt{true}, \\ y & \text{if } p = \texttt{false}, \\ x \sqcap y & \text{if } p = \perp. \end{cases}$$

The idea is that, even if the condition is undefined, the most defined partial number which is less defined than x and y can be safely produced anyway.

Lemma 3 (Elementary Axioms)

$$\begin{aligned} \mathrm{pred}_L(\mathrm{succ}_L(x)) &= x, & \mathrm{pred}_R(\mathrm{succ}_L(x)) &= 0, \\ \mathrm{pred}_L(\mathrm{succ}_R(y)) &= 1, & \mathrm{pred}_R(\mathrm{succ}_R(y)) &= y, \\ \mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) &= x \sqcap 1, & \mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) &= 0 \sqcap y, \end{aligned}$$

$$\begin{aligned} \mathrm{left} \circ \mathrm{succ}_L(x) &\sqsubseteq \texttt{true}, & \mathrm{left} \circ \mathrm{succ}_L(x) &= \bot \quad \textit{iff} \quad x \sqsubseteq 1, \\ \mathrm{left} \circ \mathrm{succ}_R(y) &\sqsubseteq \texttt{false}, & \mathrm{left} \circ \mathrm{succ}_R(y) &= \bot \quad \textit{iff} \quad y \sqsubseteq 0, \end{aligned}$$

$$x = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ \mathrm{succ}_L(\mathrm{pred}_L(x))\ \mathrm{else}\ \mathrm{succ}_R(\mathrm{pred}_R(x)).$$

Given a set X, an element $x \in X$ and a function $g : \mathbb{N} \to X$, there is a unique function $f : \mathbb{N} \to X$ such that $f(0) = x$ and $f(n+1) = g(n)$. A similar fact holds for the partial unit interval equipped with succ_L and succ_R, but we have to take into account the equation $\mathrm{succ}_L(1) = \mathrm{succ}_R(0)$.

Lemma 4 (Definition by Cases) *Let D be a domain and $g_L, g_R : \mathcal{I} \to D$ be continuous maps such that*

$$g_L(1) = g_R(0).$$

Then there is a unique continuous map $f : \mathcal{I} \to D$ such that

$$\begin{aligned} f(\mathrm{succ}_L(x)) &= g_L(x), \\ f(\mathrm{succ}_R(y)) &= g_R(y), \\ f(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) &= g_L(x) \sqcap g_R(y) \qquad \textit{if } x \sqsubseteq 1 \textit{ and } y \sqsubseteq 0, \end{aligned}$$

namely the function f defined by

$$f(x) = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ g_L(\mathrm{pred}_L(x))\ \mathrm{else}\ g_R(\mathrm{pred}_R(x)).$$

The natural numbers enjoy an induction principle, which can be expressed by saying that if a set of natural numbers contains zero and is closed under the successor operation, then it contains all natural numbers. A similar principle is enjoyed by the partial unit interval endowed with the operations succ_L and succ_R. But we have to take into account that partial real numbers are constructed by infinitely many applications of the left and right successor maps.

A subset of a domain D is called ***inductive*** if it closed under the formation of least upper bounds of directed subsets.

Lemma 5 (Dyadic Induction) *Let $A \subseteq \mathcal{I}$ be inductive, and assume that the following conditions hold:*

(i) (Base case) $\bot \in A$.
(ii) (Inductive step) $x \in A$ *and* $y \in A$ *imply*
  (a) $\mathrm{succ}_L(x) \in A$,
  (b) $\mathrm{succ}_R(y) \in A$,
  (c) $\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y) \in A$ *if* $x \sqsubseteq 1$ *and* $y \sqsubseteq 0$.

Then $A = \mathcal{I}$.

We apply this lemma in the following form:

Corollary 6 *Let D be a domain and $f, g : \mathcal{I} \to D$ be continuous maps. In order to show that $f = g$ it suffices to show that the following conditions hold:*

(i) (Base case) $f(\bot) = g(\bot)$.
(ii) (Inductive step) $f(x) = g(x)$ *and* $f(y) = g(y)$ *together imply*
  (a) $f(\mathrm{succ}_L(x)) = g(\mathrm{succ}_L(x))$,
  (b) $f(\mathrm{succ}_R(y)) = g(\mathrm{succ}_R(y))$,
  (c) $f(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) = g(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y))$ *if* $x \sqsubseteq 1$ *and* $y \sqsubseteq 0$.

Proof. If f and g are continuous then the set $A = \{x | f(x) = g(x)\}$ is inductive. □

Remark 7 *If we omit condition (c) of the inductive step, then in Lemma 5 we conclude that* $\mathrm{Max}\,\mathcal{I} \subseteq A$, *and in Corollary 6 we conclude that* $f_{|\mathrm{Max}\,\mathcal{I}} = g_{|\mathrm{Max}\,\mathcal{I}}$. *This can be used to prove that a continuous function* $\bar{f} : \mathcal{I} \to \mathcal{I}$ *is an extension of a continuous function* $f : [0,1] \to [0,1]$, *although not necessarily the* canonical *extension.* □

We can define functions on natural numbers by iteration. If X is a set, x is an element of X and $g : X \to X$ is a function, then there is a unique function $f : \mathbb{N} \to X$ such that $f(0) = x$ and $f(n+1) = g(f(n))$. A similar fact holds for the partial unit interval equipped with succ_L and succ_R. We first need a lemma:

Lemma 8 *Let D be a domain, $g_L, g_R : D \to D$ be continuous maps, and $f : \mathcal{I} \to D$ be any continuous solution to the functional equation*

$$f(x) = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ g_L(f(\mathrm{pred}_L(x)))\ \mathrm{else}\ g_R(f(\mathrm{pred}_R(x))).$$

Then the following statements hold:

(i) $f(0)$, $f(1)$ *and* $f(\bot)$ *are fixed points of* g_L, g_R *and* $g_L \sqcap g_R$ *respectively.*
(ii) f *is uniquely determined by the values that it assumes at* 0, 1, *and* $\bot$.

(iii) *f is the least continuous solution iff $f(0) = \mathrm{fix}(g_L)$, $f(1) = \mathrm{fix}(g_L)$, and $f(\bot) = \mathrm{fix}(g_L \sqcap g_R)$.*

In particular, if g_L, g_R, and $g_L \sqcap g_R$ have unique fixed points then there is a unique continuous solution.

Lemma 9 (Dyadic Iteration) *Let D be a domain, and $g_L, g_R : D \to D$ be continuous maps such that*

$$g_L(\mathrm{fix}(g_R)) = g_R(\mathrm{fix}(g_L)).$$

Then there is a unique continuous map $f : \mathcal{I} \to D$ satisfying the equations

(Base case)

$$\begin{aligned} f(0) &= \mathrm{fix}(g_L) \\ f(1) &= \mathrm{fix}(g_R) \\ f(\bot) &= \mathrm{fix}(g_L \sqcap g_R) \end{aligned}$$

(Iteration step)

$$\begin{aligned} f(\mathrm{succ}_L(x)) &= g_L(f(x)) \\ f(\mathrm{succ}_R(y)) &= g_R(f(y)) \\ f(\mathrm{succ}_R(x) \sqcap \mathrm{succ}_R(y)) &= g_L(f(x)) \sqcap g_R(f(y)) \qquad \text{if } x \sqsubseteq 1 \text{ and } y \sqsubseteq 0, \end{aligned}$$

namely the least continuous solution to the equation

$$f(x) = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ g_L(f(\mathrm{pred}_L(x)))\ \mathrm{else}\ g_R(f(\mathrm{pred}_R(x))).$$

Finally, the natural numbers system is uniquely specified, up to isomorphism, by the so-called Peano axioms, which are essentially the properties that we informally considered above for the sake of motivation. This idea is made formal in e.g. Stoll [35], where *unary systems* are used as a tool (a unary system is a set X together with an element $x \in X$ and a function $s : X \to X$).

In the following definition, the domain D generalizes the partial unit interval and the maps a_L and a_R generalize the constructor maps succ_L and succ_R respectively.

Definition 10 *A **binary system** is a domain D equipped with a pair of continuous maps $a_L, a_R : D \to D$ such that*

$$a_L(1) = a_R(0),$$

where

$$0 \stackrel{\text{def}}{=} \text{fix}(a_L), \qquad 1 \stackrel{\text{def}}{=} \text{fix}(a_R).$$

We also impose the technical condition $\text{fix}(a_L \sqcap a_R) = \bot$*, which ensures that homomorphisms defined below, as Lemma 9 suggests, make binary systems into a category under ordinary function composition.*

A ***homomorphism*** *from a binary system* (D, a_L, a_R) *to a binary system* (E, b_L, b_R) *is a continuous map* $f : D \to E$ *such that*

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(\bot) &= \bot \\ f(a_L(x)) &= b_L(f(x)) \\ f(a_R(y)) &= b_R(f(y)) \\ f(a_L(x) \sqcap a_R(y)) &= b_L(f(x)) \sqcap b_R(f(y)) \qquad \textit{if } x \sqsubseteq 1 \textit{ and } y \sqsubseteq 0. \end{aligned}$$

□

Binary systems were introduced and investigated in the context of uniform spaces in the extended abstract [8]. Lemma 9 can be formulated as

Theorem 11 $(\mathcal{I}, \text{succ}_L, \text{succ}_R)$ *is an initial object in the category of binary systems.*

By this we mean, of course, that there is a unique homomorphism from it to any other binary system. Since any two initial objects are isomorphic, this together with the following theorem axiomatically characterize the binary system $(\mathcal{I}, \text{succ}_L, \text{succ}_R)$ up to isomorphism, without explicit reference to real numbers or intervals:

Theorem 12 *A binary system* (D, s_L, s_R) *is initial iff the following conditions hold:*

(i) 0 *and* 1 *are the unique fixed points of* s_L *and* s_R*.*
(ii) $0 \neq 1$ *and* $0 \sqcap 1 = \bot$*.*
(iii) There are continuous maps $l : D \to \mathcal{B}$ *and* $p_L, p_R : D \to D$ *such that*

$$\begin{aligned} p_L(s_L(x)) &= x, & p_R(s_L(x)) &= 0, \\ p_L(s_R(y)) &= 1, & p_R(s_R(y)) &= y, \\ p_L(s_L(x) \sqcap s_R(y)) &= x \sqcap 1, & p_R(s_L(x) \sqcap s_R(y)) &= 0 \sqcap y, \end{aligned}$$

$$\begin{aligned} l(s_L(x)) &\sqsubseteq \texttt{true}, & l(s_L(x)) &= \bot \textit{ iff } x \sqsubseteq 1, \\ l(s_R(y)) &\sqsubseteq \texttt{false}, & l(s_R(y)) &= \bot \textit{ iff } y \sqsubseteq 0, \end{aligned}$$

$$x = \text{pif } l(x) \text{ then } s_L(p_L(x)) \text{ else } s_R(p_R(x)).$$

(iv) For any inductive set $A \subseteq D$, if
(a) $\bot \in A$,
(b) $x \in A$ and $y \in A$ imply
$s_L(x) \in A$,
$s_R(y) \in A$,
$s_L(x) \sqcap s_R(y) \in A$ *if* $x \sqsubseteq 1$ *and* $y \sqsubseteq 0$,
then $A = D$.

This and the following lemma are abstract versions of Lemmas 3 and 4.

Lemma 13 (Existence of Destructors) *Let $D = (D, s_L, s_R)$ be a binary system. Then there is at most one triple of continuous maps $l : D \to \mathcal{B}$ and $p_L, p_R : D \to D$ satisfying the conditions of Theorem 12(iii).*

Moreover, in this case D admits definition by cases, in the sense that for each domain E and each pair of continuous maps $g_L, g_L : D \to E$ with

$$g_L(1) = g_R(0)$$

there is a unique continuous map $f : D \to E$ such that

$$\begin{aligned} f(s_L(x)) &= g_L(x) \\ f(s_R(y)) &= g_R(y) \\ f(s_L(x) \sqcap s_R(y)) &= g_L(x) \sqcap g_R(y) \qquad \text{if } x \sqsubseteq 1 \text{ and } y \sqsubseteq 0. \end{aligned}$$

Proof. of Theorem 12. The previous lemmas show that the initial binary system satisfies the conditions. Conversely, the proof of Lemma 9 uses only the properties of binary systems in the statement of the theorem, without mentioning any particular property of $(\mathcal{I}, \mathrm{succ}_L, \mathrm{succ}_R)$, except for the equations

$$\begin{aligned} \mathrm{left}(0) &= \texttt{true}, & \mathrm{left}(1) &= \texttt{false}, \\ \mathrm{pred}_L(0) &= 0, & \mathrm{pred}_R(1) &= 1, \\ \mathrm{pred}_L(\bot) &= \bot, & \mathrm{pred}_R(\bot) &= \bot, \end{aligned}$$

indirectly in Lemma 8, which easily follow from the other conditions. □

We finish this section with the proofs of the lemmas used to establish Theorems 11 and 12.

Proof. of Lemma 1 (Dyadic Trichotomy): (i): succ_L is bijective onto its image. Hence there is at most one y with $x = \mathrm{succ}_L(y)$. The image of succ_L is $\uparrow L$

by definition. But $x \leq 1/2$ iff $L \sqsubseteq x$. (ii): Similar. (iii): In this case $1/2 \in x$. Hence $x = [\underline{x}, 1/2] \sqcap [1/2, \overline{x}] = \mathrm{succ}_L([2\underline{x}, 1]) \sqcap \mathrm{succ}_L([0, 2\overline{x} - 1])$. $\square$

Proof. of Lemma 3 (Elementary Axioms): Routine verification, included for the sake of completeness. The equation $\mathrm{pred}_a \circ \mathrm{succ}_a = \mathrm{id}$ holds by construction. Also,

$$\begin{aligned}\mathrm{pred}_L(\mathrm{succ}_R(y)) &= \min(2((y+1)/2), 1) = \min(y+1, 1) = 1,\\ \mathrm{pred}_R(\mathrm{succ}_L(x)) &= \max(0, 2(x/2) - 1) = \max(0, x-1) = 0.\end{aligned}$$

Since for all $p, q \in [0,1]$ we have that $p/2 \leq (q+1)/2$, as $p/2 \in L$ and $(q+1)/2 \in R$, it follows that

$$\begin{aligned}\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y) &= [\underline{x}/2, \overline{x}/2] \sqcap [(\underline{y}+1)/2, (\overline{y}+1)/2]\\ &= [\underline{x}/2, (\overline{y}+1)/2].\end{aligned}$$

Hence $\mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) = \mathrm{pred}_L([\underline{x}/2, (\overline{y}+1)/2]) = [\underline{x}, 1] = x \sqcap 1$. Similarly, one has that $\mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) = 0 \sqcap y$. The statements about left follow from the fact that $\mathrm{succ}_L(x) \leq 1/2$ and that $\mathrm{succ}_L(x) < 1/2$ iff $x/2 < 1/2$ iff $x < 1$ iff $x \not\sqsubseteq 1$. Similarly, $\mathrm{succ}_R(x) \geq 1/2$, and $\mathrm{succ}_R(y) < 1/2$ iff $y \not\sqsubseteq 0$. For the last equation, only the case $\mathrm{left}(x) = \bot$ is not immediate. In this case $x \uparrow 1/2$, which is equivalent to $x \sqsubseteq 1/2$ as $1/2$ is maximal, and means that $1/2 \in x$. Hence

$$\begin{aligned}x &= [\underline{x}, 1/2] \sqcap [1/2, \overline{x}]\\ &= \mathrm{succ}_L(\mathrm{pred}_L(x)) \sqcap \mathrm{succ}_R(\mathrm{pred}_R(x))\\ &= \text{pif } \mathrm{left}(x) \text{ then } \mathrm{succ}_L(\mathrm{pred}_L(x)) \text{ else } \mathrm{succ}_R(\mathrm{pred}_R(x)).\end{aligned}$$ $\square$

Proof. of Lemma 4 (Definition by cases): It is clear from Lemma 1 that there is at most one such function. We show that f as defined in the statement of the lemma is such a function. If $x \not\sqsubseteq 1$ then $\mathrm{left}(\mathrm{succ}_L(x)) = \texttt{true}$ and

$$f(\mathrm{succ}_L(x)) = g_L(\mathrm{pred}_L(\mathrm{succ}_L(x))) = g_L(x).$$

Otherwise $\mathrm{left}(\mathrm{succ}_L(x)) = \bot$ and

$$\begin{aligned}f(\mathrm{succ}_L(x)) &= g_L(\mathrm{pred}_L(\mathrm{succ}_L(x))) \sqcap g_R(\mathrm{pred}_R(\mathrm{succ}_L(x)))\\ &= g_L(x) \sqcap g_R(0) = g_L(x) \sqcap g_L(1) = g_L(x),\end{aligned}$$

because $g_L(x) \sqsubseteq g_L(1)$, by monotonicity. Similarly, a case analysis on y shows that the equation $f(\mathrm{succ}_R(y)) = g_R(y)$ holds. Assume that $x \sqsubseteq 1$ and $y \sqsubseteq 0$. Then

$$\begin{aligned}&f(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y))\\ &= g_L(\mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y))) \sqcap g_R(\mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))\\ &= g_L(x \sqcap 1) \sqcap g_R(0 \sqcap y) = g_L(x) \sqcap g_R(y),\end{aligned}$$

which concludes the proof. □

Proof. of Lemma 5 (Dyadic Induction): Let B be the basis of $\mathcal{I}$ consisting of intervals with *distinct* dyadic end-points. In order to show that $A = \mathcal{I}$, it suffices to conclude that $B \subseteq A$, because B is a basis of $\mathcal{I}$. But since

$$B = \bigcup_n B_n, \quad \text{where} \quad B_n = \{[l/2^n, m/2^n] | 0 \leq l < m \leq 2^n\},$$

it suffices to show that $B_n \subseteq A$ for all n by induction on n. For $n = 0$ this is immediate because $B_0 = \{\bot\}$ and $\bot \in A$ by hypothesis. Assume that $B_n \subseteq A$, and define

$$\begin{aligned}
L_n &= \mathrm{succ}_L(B_n), \\
R_n &= \mathrm{succ}_R(B_n), \\
C_n &= \{\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y) | x, y \in B_n \wedge x \sqsubseteq 1 \wedge y \sqsubseteq 0\}.
\end{aligned}$$

Then $L_n \subseteq \mathrm{succ}_L(A) \subseteq A$ because $x \in A$ implies $\mathrm{succ}_L(x) \in A$ by hypothesis. Similarly, $R_n \subseteq A$ and $C_n \subseteq A$. Hence $L_n \cup R_n \cup C_n \subseteq A$. But

$$\begin{aligned}
&L_n \cup R_n \cup C_n \\
&= \left\{ \left[\frac{l}{2^{n+1}}, \frac{m}{2^{n+1}} \right] \middle| 0 \leq l < m \leq 2^n \right\} \cup \left\{ \left[\frac{l+2^n}{2^{n+1}}, \frac{m+2^n}{2^{n+1}} \right] \middle| 0 \leq l < m \leq 2^n \right\} \\
&\cup \left\{ \left[\frac{l}{2^{n+1}}, \frac{1}{2} \right] \sqcap \left[\frac{1}{2}, \frac{m+2^n}{2^{n+1}} \right] \middle| 0 \leq l < 2^n \wedge 0 < m \leq 2^n \right\} \\
&= \left\{ \left[\frac{l}{2^{n+1}}, \frac{m}{2^{n+1}} \right] \middle| 0 \leq l < m \leq 2^n \right\} \cup \left\{ \left[\frac{l}{2^{n+1}}, \frac{m}{2^{n+1}} \right] \middle| 2^n \leq l < m \leq 2^{n+1} \right\} \\
&\cup \left\{ \left[\frac{l}{2^{n+1}}, \frac{m}{2^{n+1}} \right] \middle| 0 \leq l < 2^n < m \leq 2^{n+1} \right\} \\
&= \left\{ \left[\frac{l}{2^{n+1}}, \frac{m}{2^{n+1}} \right] \middle| 0 \leq l < m \leq 2^{n+1} \right\} \\
&= B_{n+1}.
\end{aligned}$$

Therefore $B_{n+1} \subseteq A$, which concludes the inductive argument. □

Proof. of Lemma 8: (i): One has that $f(0) = g_L(f(0))$ because $\mathrm{left}(0) = \texttt{true}$ and $\mathrm{pred}_L(0) = 0$. Similarly, $f(1) = g_R(f(1))$. Since $\mathrm{left}(\bot) = \bot$ and $\mathrm{pred}_L(\bot) = \mathrm{pred}_R(\bot)$, we have that $f(\bot) = g_L(f(\bot)) \sqcap g_R(f(\bot)) = (g_L \sqcap g_R)(f(\bot))$.

(ii): We show by dyadic induction as in Corollary 6 that if f' is a continuous solution agreeing with f at 0, 1 and $\bot$ then $f = f'$. (Base case): By hypothesis. (Inductive step): Assume that $f(x) = f'(x)$ and $f(y) = f'(y)$. If $\mathrm{left} \circ \mathrm{succ}_L(x) = \texttt{true}$ then

$$\begin{aligned} f(\mathrm{succ}_L(x)) &= g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x)))) = g_L(f(x)) \\ &= g_L(f'(x)) = g_L(f'(\mathrm{pred}_L(\mathrm{succ}_L(x)))) = f'(\mathrm{succ}_L(x)). \end{aligned}$$

Otherwise $\mathrm{left}(\mathrm{succ}_L(x)) = \bot$ and $x \sqsubseteq 1$. Therefore,

$$\begin{aligned} f(\mathrm{succ}_L(x)) &= g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x)))) \sqcap g_R(f(\mathrm{pred}_R(\mathrm{succ}_L(x)))) \\ &= g_L(f(x)) \sqcap g_R(f(0)) = g_L(f'(x)) \sqcap g_R(f'(0)) \\ &= g_L(f'(\mathrm{pred}_L(\mathrm{succ}_L(x)))) \sqcap g_R(f'(\mathrm{pred}_R(\mathrm{succ}_L(x)))) \\ &= f'(\mathrm{succ}_L(x)). \end{aligned}$$

Similarly, $f(\mathrm{succ}_R(y)) = f'(\mathrm{succ}_R(y))$. Assume that $x \sqsubseteq 1$ and $y \sqsubseteq 0$. Then

$$\begin{aligned} &f(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) \\ &= g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \sqcap g_R(f(\mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \\ &= g_L(f(x \sqcap 1)) \sqcap g_R(f(0 \sqcap y)) = g_L(f(x)) \sqcap g_R(f(y)) \\ &= g_L(f'(x)) \sqcap g_R(f'(y)) = g_L(f'(x \sqcap 1)) \sqcap g_R(f'(0 \sqcap y)) \\ &= g_L(f'(\mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \sqcap g_R(f'(\mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \\ &= f'(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)). \end{aligned}$$

(iii): In view of (i) and (ii), it suffices to show that if f is the least solution then it satisfies the condition. Assume that f is the least solution. Then $f = \bigsqcup_n f_n$ where

$$\begin{aligned} f_0(x) &= \bot \\ f_{n+1}(x) &= \text{pif } \mathrm{left}(x) \text{ then } g_L(f_n(\mathrm{pred}_L(x))) \text{ else } g_R(f_n(\mathrm{pred}_R(x))). \end{aligned}$$

But $f_n(0) = g_L^n(\bot)$ because $\mathrm{left}(0) = \texttt{true}$ and $\mathrm{pred}_L(0) = 0$. Hence $f(0) = \mathrm{fix}(g_L)$. Similarly, $f(1) = \mathrm{fix}(g_R)$. Also, $f_{n+1}(\bot) = (g_L \sqcap g_R)^n(\bot)$, because

$$f_{n+1}(\bot) = g_L(f_n(\bot)) \sqcap g_R(f_n(\bot)) = (g_L \sqcap g_R)(f_n(\bot)).$$

Therefore $f(\bot) = \mathrm{fix}(g_L \sqcap g_R)$. $\square$

Proof. of Lemma 9 (Dyadic Iteration): Define $F : [\mathcal{I} \to D] \to [\mathcal{I} \to D]$ by

$$F(f)(x) = \text{pif } \mathrm{left}(x) \text{ then } g_L(f(\mathrm{pred}_L(x))) \text{ else } g_R(f(\mathrm{pred}_R(x))),$$

and let $f : \mathcal{I} \to D$ be a continuous map satisfying the base case. By Lemma 8, it suffices to show that f satisfies the recursion step iff $f = F(f)$. But, by Lemma 4, this is equivalent to show that

$$\begin{aligned} F(f)(\mathrm{succ}_L(x)) &= g_L(f(x)) \\ F(f)(\mathrm{succ}_R(y)) &= g_R(f(y)) \\ F(f)(\mathrm{succ}_R(x) \sqcap \mathrm{succ}_R(y)) &= g_L(f(x)) \sqcap g_R(f(y)) \qquad \text{if } x \sqsubseteq 1 \text{ and } y \sqsubseteq 0. \end{aligned}$$

In fact, assuming that these equations hold, if $f = F(f)$ then f satisfies the recursion step; and, conversely, if f satisfies the recursion step both f and $F(f)$ satisfy the same definition by cases, and therefore they have to be the same by Lemma 4. If $x \not\sqsubseteq 1$ then $\mathrm{left}(\mathrm{succ}_L(x)) = \mathtt{true}$ and

$$F(f)(\mathrm{succ}_L(x)) = g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x)))) = g_L(f(x)).$$

Otherwise $\mathrm{left}(\mathrm{succ}_L(x)) = \bot$ and

$$\begin{aligned} F(f)(\mathrm{succ}_L(x)) &= g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x)))) \sqcap g_R(f(\mathrm{pred}_R(\mathrm{succ}_L(x)))) \\ &= g_L(f(x)) \sqcap g_R(f(0)) = g_L(f(x)) \sqcap g_R(\mathrm{fix}(g_L)) \\ &= g_L(f(x)) \sqcap g_L(\mathrm{fix}(g_R)) = g_L(f(x)) \sqcap g_L(f(1)) \\ &= g_L(f(x)), \end{aligned}$$

because $g_L(f(x)) \sqsubseteq g_L(f(1))$, by monotonicity. Similarly, $F(f)(\mathrm{succ}_R(y)) = g_R(f(y))$. Assume that $x \sqsubseteq 1$ and $y \sqsubseteq 0$. Then

$$\begin{aligned} &F(f)(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)) \\ &= g_L(f(\mathrm{pred}_L(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \sqcap g_R(f(\mathrm{pred}_R(\mathrm{succ}_L(x) \sqcap \mathrm{succ}_R(y)))) \\ &= g_L(f(x \sqcap 1)) \sqcap g_R(f(0 \sqcap y)) = g_L(f(x)) \sqcap g_R(f(y)). \qquad \square \end{aligned}$$

Proof. of Lemma 13 (Existence of Destructors): Assume that such maps exist. Claim: For every $x \in \mathcal{I}$,

(i) if $l(x) = \mathtt{true}$ then $x = s_L(y)$ for a unique y,
(ii) if $l(x) = \mathtt{false}$ then $x = s_R(z)$ for a unique z,
(iii) if $l(x) = \bot$ then $x = s_L(y) \sqcap s_R(z)$ for unique $y \sqsubseteq 1$ and $z \sqsubseteq 0$.

(i): There is at most one such y because p_L is a left inverse of s_L. Since

$$x = \mathrm{pif}\ \mathtt{true}\ \mathrm{then}\ s_L(p_L(x))\ \mathrm{else}\ s_R(p_R(x)) = s_L(p_L(x)),$$

We can take $y = p_L(x)$. (ii): Similar. (iii): (Uniqueness) Assume that $s_L(y) \sqcap s_R(z) = s_L(y') \sqcap s_R(z')$ for $y, y' \sqsubseteq 1$ and $z, z' \sqsubseteq 0$. Then, by applying p_L to both sides, we obtain $y \sqcap 1 = y' \sqcap 1$. But $y \sqcap 1 = y$ and $y' \sqcap 1 = y'$. Therefore $y = y'$. Similarly, $z = z'$. (Existence):

$$x = \mathrm{pif}\ \bot\ \mathrm{then}\ s_L(p_L(x))\ \mathrm{else}\ s_R(p_R(x)) = s_L(p_L(x) \sqcap s_R(p_R(x))).$$

Hence $p_L(x) = p_L(x) \sqcap 1$ and $p_R(x) = 0 \sqcap p_R(x)$. We can thus let $y = p_L(x)$ and $z = p_R(x)$, and the proof of the claim is concluded.

It follows that there is at most one f satisfying the definition-by-cases scheme. Therefore there is at most one triple of maps as specified above, because p_L

and p_R satisfy the definition-by-cases scheme, and h is completely specified by the above clauses by virtue of the claim (the inequality $l(s_L(x) \sqcap s_R(y)) \sqsubseteq \bot$ holds by monotonicity). Finally, a function f satisfying the definition-by-cases scheme can be constructed as in Lemma 4, because it only uses the abstract properties of succ_L, succ_R, left, pred_L, and pred_R considered in the statement of the present lemma (and proved in Lemma 3). □

4 Generalized domain equations

(At this point the reader is assumed to be familiar with the theory of domain equations recalled in Section 1.4.)

We introduce a new technique, based on the theory of domain equations, which allows us to handle structural recursion with hypotheses that are weaker than is usual in domain theory. The basic idea is to consider not a distinguished solution D for a domain equation $D \cong \mathbf{F}D$, but more generally a domain D which is a retract of $\mathbf{F}D$ in a special way. We refer to such a domain D as an $\mathbf{F}$-inductive retract.

The constructor and destructor maps for the partial unit interval discussed in Section 3 form an inductive retraction, as it is shown in Section 4.2, which is the object of study of Section 5. In this section we develop the general theory of inductive retracts.

4.1 Inductive retracts

Through this section $\mathbf{F}$ is an endofunctor of a category $\mathbf{X}$.

Definition 14 *An* $\mathbf{F}$***-inductive retract*** *is an object X together with an algebra $\kappa : \mathbf{F}X \to X$ and a coalgebra $\delta : X \to \mathbf{F}X$, subject to the condition*

$$f = \kappa \circ \mathbf{F}f \circ \delta \text{ iff } f = \mathrm{id}_X. \quad \square$$

The arrows κ and δ can be thought as a constructor and a destructor maps respectively. The first equation of the definition is illustrated in the following diagram:

$$\begin{array}{ccc} \mathbf{F}X & \xrightarrow{\kappa} & X \\ {\scriptstyle \mathbf{F}f}\uparrow & & \downarrow{\scriptstyle f} \\ \mathbf{F}X & \xleftarrow[\delta]{} & X \end{array}$$

The right-to-left implication shows that

$$\kappa \circ \delta = \mathrm{id}_X$$

and hence X is a retract of $\mathbf{F}X$. Also, notice that if $\langle \kappa, \delta \rangle$ is an $\mathbf{F}$-inductive retraction in $\mathbf{X}$, then $\langle \delta, \kappa \rangle$ is an $\mathbf{F}$-inductive retraction in the opposite category $\mathbf{X}^{\mathrm{op}}$.

4.2 An example: the partial unit interval

In order to obtain an inductive retraction for the partial unit interval, we put the constructors (respectively destructors) together.

Define a functor $\mathbf{T} : \mathbf{SDom} \to \mathbf{SDom}$ by

$$\mathbf{T}D = \mathcal{B} \times D \times D, \qquad \mathbf{T}f = \mathrm{id}_{\mathcal{B}} \times f \times f,$$

and define and algebra $\mathrm{constr} : \mathbf{T}\mathcal{I} \to \mathcal{I}$ and an algebra $\mathrm{destr} : \mathcal{I} \to \mathbf{T}\mathcal{I}$ by

$$\begin{aligned} \mathrm{constr} &= \mathrm{pif} \circ (\mathrm{id}_{\mathcal{B}} \times \mathrm{succ}_L \times \mathrm{succ}_R), \\ \mathrm{destr} &= \langle \mathrm{left}, \mathrm{pred}_L, \mathrm{pred}_R \rangle. \end{aligned}$$

That is,

$$\begin{aligned} \mathrm{constr}(p, y, z) &= \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}_L(y)\ \mathrm{else}\ \mathrm{succ}_R(z), \\ \mathrm{destr}(x) &= \langle \mathrm{left}(x), \mathrm{pred}_L(x), \mathrm{pred}_R(x) \rangle. \end{aligned}$$

Proposition 15 *The algebra* $\mathrm{constr} : \mathbf{T}\mathcal{I} \to \mathcal{I}$ *and the coalgebra* $\mathrm{destr} : \mathcal{I} \to \mathbf{T}\mathcal{I}$ *together form a* $\mathbf{T}$*-inductive retraction.*

Proof. The equation $f = \mathrm{constr} \circ \mathbf{T}f \circ \mathrm{destr}$ is equivalent to the equation

$$f(x) = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ \mathrm{succ}_L(f(\mathrm{pred}_L(x)))\ \mathrm{else}\ \mathrm{succ}_R(f(\mathrm{pred}_R(x))).$$

We know that $f = \mathrm{id}_{\mathcal{I}}$ is a solution by Lemma 3. But succ_L and succ_R and $\mathrm{succ}_R \sqcap \mathrm{succ}_L$ have unique fixed-points. Therefore this is the unique solution by virtue of Lemma 8. $\square$

4.3 Inductive retractions and bifree algebras

The following proposition shows that inductive retractions generalize bifree algebras (cf. Section 1.4):

Proposition 16 *Let $\delta : X \leftrightarrows \mathbf{F}X : \kappa$ be an $\mathbf{F}$-inductive isomorphism. If $\mathbf{F}$ has a bifree algebra then it is isomorphic to κ.*

Proof. Let $i : \mathbf{F}C \to C$ be a bifree algebra, $r : i \to \kappa$ be the unique algebra homomorphism and $s : \delta \to i^{-1}$ be the unique coalgebra homomorphism. This means that $r \circ i = \kappa \circ \mathbf{F}r$ and $i^{-1} \circ s = \mathbf{F}s \circ \delta$. Hence

$$r \circ s = r \circ i \circ i^{-1} \circ s = \kappa \circ \mathbf{F}r \circ \mathbf{F}s \circ \delta = \kappa \circ \mathbf{F}(r \circ s) \circ \delta.$$

By inductivity, $r \circ s = \mathrm{id}_X$. Since $s = i \circ \mathbf{F}s \circ \delta$, we have that

$$s \circ r \circ i = i \circ \mathbf{F}s \circ \delta \circ \kappa \circ \mathbf{F}r = i \circ \mathbf{F}(s \circ r).$$

Hence $s \circ r : i \to i$ and therefore $s \circ r = \mathrm{id}_C$. □

Throughout the remainder of this section, $i : \mathbf{F}C \to C$ is an arbitrary bifree algebra and $\delta : X \leftrightarrows \mathbf{F}X : \kappa$ is an arbitrary $\mathbf{F}$-inductive retraction.

The first part of the proof of Proposition 16 shows that every inductive retract is a retract of the bifree algebra, in a canonical way:

Lemma 17 *If $r : i \to \kappa$ and $s : \delta \to i^{-1}$ are the unique (co)algebra homomorphisms then the arrows $s : X \leftrightarrows C : r$ form a retraction with $r \circ s = \mathrm{id}_X$.*

4.4 Structural recursion and corecursion

Proposition 18 *Let $r : i \to \kappa$, $s : \delta \to i^{-1}$ and $e = s \circ r : C \to C$.*

(i) For any algebra $a : \mathbf{F}A \to A$, there is a homomorphism $f : \kappa \to a$ iff $h = h \circ e$, where $h : i \to a$, and in this case $f = h \circ s$.
(ii) For any coalgebra $b : B \to \mathbf{F}B$, there is a homomorphism $g : b \to \delta$ iff $k = e \circ k$, where $k : b \to i^{-1}$, and in this case $g = r \circ k$.

Proof. (i): If $f : \kappa \to a$ then $f \circ r = h$ because $r : i \to \kappa$. Therefore $f = h \circ s$ and $h = h \circ s \circ r$. Conversely, if $f \circ r : i \to a$ then $f : \kappa \to a$ because

$$\begin{aligned} f \circ \kappa &= f \circ \kappa \circ \mathbf{F}(r \circ s) = f \circ \kappa \circ \mathbf{F}r \circ \mathbf{F}s = f \circ r \circ i \circ \mathbf{F}s \\ &= a \circ \mathbf{F}(f \circ r) \circ \mathbf{F}s = a \circ \mathbf{F}f \circ \mathbf{F}(r \circ s) = a \circ \mathbf{F}f. \end{aligned}$$

If $h \circ s \circ r = h$ then this holds in particular for $f = h \circ s$. (ii): Dual to (i). □

Condition (i) means that h respects the congruence on C induced by the idempotent $e = s \circ r$, and that f is the restriction of h to X via s. Dually, condition (ii) means that the image of k is contained in image of e and that g is the corestriction of k to X via r.

Corollary 19

(i) For every algebra $a : \mathbf{F}A \to A$ there is at most one homomorphism $f : \kappa \to a$.
(ii) For every coalgebra $b : B \to \mathbf{F}B$ there is at most one homomorphism $g : b \to \delta$.

Only for the last result of this subsection, we assume that our base category $\mathbf{X}$ is the category **SDom** of domains and strict continuous maps.

Proposition 20 *Let $\mathbf{F} : \mathbf{SDom} \to \mathbf{SDom}$ be locally continuous.*

(i) If there is a homomorphism $f : \kappa \to a$ for a given algebra $a : \mathbf{F}A \to A$ then it can be recursively defined by $f = a \circ \mathbf{F}f \circ \delta$.
(ii) If there is a homomorphism $g : b \to \delta$ for a given coalgebra $b : B \to \mathbf{F}B$ then it can be recursively defined by $g = \kappa \circ \mathbf{F}g \circ b$.

Proof. (i): The least solution of the equation is $f' = \bigsqcup_n f_n$, where the sequence f_n is inductively defined by $f_0 = \bot$ and $f_{n+1} = a \circ \mathbf{F}f_n \circ \delta$. Define $\mathrm{id}_n : X \to X$ by $\mathrm{id}_0 = \bot$ and $\mathrm{id}_{n+1} = \kappa \circ \mathbf{F}\mathrm{id}_n \circ \delta$. By local continuity of $\mathbf{F}$,

$$\bigsqcup_n \mathrm{id}_n = \bigsqcup_n \mathrm{id}_{n+1} = \bigsqcup_n (\kappa \circ \mathbf{F}\mathrm{id}_n \circ \delta) = \kappa \circ \mathbf{F}\left(\bigsqcup_n \mathrm{id}_n\right) \circ \delta.$$

Hence $\bigsqcup_n \mathrm{id}_n = \mathrm{id}_X$ by inductivity. Since f is strict, we have that $f_0 = f \circ \mathrm{id}_0$. Assuming that $f_n = f \circ \mathrm{id}_n$ we deduce that

$$f_{n+1} = a \circ \mathbf{F}f_n \circ \delta = a \circ \mathbf{F}f \circ \mathbf{F}\mathrm{id}_n \circ \delta = f \circ \kappa \circ \mathbf{F}\mathrm{id}_n \circ \delta = f \circ \mathrm{id}_{n+1}.$$

Hence $f_n = f \circ \mathrm{id}_n$ for every n. Therefore

$$f' = \bigsqcup f_n = \bigsqcup_n (f \circ \mathrm{id}_n) = f \circ \bigsqcup_n \mathrm{id}_n = f \circ \mathrm{id}_X = f.$$

(ii): Dual to (i). □

Thus, in order to find a recursive definition of a function $f : X \to A$ we can try to find an algebra $a : \mathbf{F}A \to A$ such that $f : \kappa \to a$ is a homomorphism, and in order to find a recursive definition of a function $g : B \to X$ we can try to find a coalgebra $b : B \to \mathbf{F}B$ such that $g : b \to \delta$ is a homomorphism. If we

succeed in finding such algebra a and coalgebra b, then we obtain a definition of f by *structural recursion* and a definition of g by *structural corecursion.*

4.5 Biquotients of bifree algebras

We have seen that any $\mathbf{F}$-inductive retraction $\delta : \mathbf{X} \leftrightarrows \mathbf{FX} : \kappa$ appears as a retract of the bifree $\mathbf{F}$-algebra $i : \mathbf{F}C \to C$ via $r : i \to \kappa$ and $s : \delta \to i^{-1}$ with $r \circ s = \mathrm{id}_X$.

In this subsection, which is not needed in the development that follows, we characterize for a bifree algebra $i : \mathbf{F}C \to C$ the idempotents $e : C \to C$ which admit a splitting $e = s \circ r$ of the kind just described. Recall that any idempotent in **SDom** splits through its image. But notice that we are still working in an arbitrary category $\mathbf{X}$.

Definition 21 *Let $e : C \to C$ be an idempotent and define an algebra $a : \mathbf{F}C \to C$ and a coalgebra $b : C \to \mathbf{F}C$ by*

$$a = e \circ i, \qquad b = i^{-1} \circ e.$$

*We say that e is a **biquotient** of the bifree algebra $i : \mathbf{F}C \to C$ if the following conditions hold:*

(B1) $e : i \to a$,
(B2) $e : b \to i^{-1}$,
(B3) $h = a \circ \mathbf{F}h \circ b$ *iff* $h = e$. $\square$

Theorem 22

(i) If $\delta : X \rightleftarrows \mathbf{F}X : \kappa$ is an $\mathbf{F}$-inductive retraction, $r : i \to \kappa$ and $s : \delta \to i^{-1}$, then $e \stackrel{\text{def}}{=} s \circ r$ is a biquotient of i. Moreover, κ and δ can be recovered from r and s as

$$\kappa = r \circ i \circ \mathbf{F}s \qquad \delta = \mathbf{F}r \circ i^{-1} \circ s.$$

(ii) If $e : C \to C$ is a biquotient of i and $e = s \circ r$ with $r \circ s = id_X$, then the maps

$$\kappa \stackrel{\text{def}}{=} r \circ i \circ \mathbf{F}s : \mathbf{F}X \to X$$
$$\delta \stackrel{\text{def}}{=} \mathbf{F}r \circ i^{-1} \circ s : X \to \mathbf{F}X$$

constitute an $\mathbf{F}$-inductive retraction. Moreover, we have $r : i \to \kappa$ and $s : \delta \to i^{-1}$.

Proof. (i): Conditions (B1) and (B2) hold by the following equational reasoning:

$$\begin{aligned} e \circ i \circ \mathbf{F}e &= s \circ r \circ i \circ \mathbf{F}s \circ \mathbf{F}r = s \circ \kappa \circ \mathbf{F}r \circ \mathbf{F}s \circ \mathbf{F}r \\ &= s \circ \kappa \circ \mathbf{F}r = s \circ r \circ i = e \circ i, \\ \mathbf{F}e \circ i^{-1} \circ e &= \mathbf{F}s \circ \mathbf{F}r \circ i^{-1} \circ s \circ r = \mathbf{F}s \circ \mathbf{F}r \circ \mathbf{F}s \circ \delta \circ r \\ &= \mathbf{F}s \circ \delta \circ r = i^{-1} \circ s \circ r = i^{-1} \circ e. \end{aligned}$$

From this we immediately obtain

$$e \circ i \circ \mathbf{F}e \circ i^{-1} \circ e = e \circ i \circ i^{-1} \circ e = e \circ e = e.$$

For the other implication of condition (B3), let $h : C \to C$ with $e \circ i \circ \mathbf{F}h \circ i^{-1} \circ e = h$. It follows that

$$r \circ i \circ \mathbf{F}h \circ i^{-1} \circ s = r \circ h \circ s.$$

Hence

$$\begin{aligned} r \circ h \circ s &= r \circ i \circ \mathbf{F}h \circ i^{-1} \circ s = \kappa \circ \mathbf{F}r \circ \mathbf{F}h \circ \mathbf{F}s \circ \delta \\ &= \kappa \circ \mathbf{F}(r \circ h \circ s) \circ \delta, \end{aligned}$$

which entails $r \circ h \circ s = \mathrm{id}_X$ as κ and δ form an $\mathbf{F}$-inductive retraction. Thus we get

$$h = e \circ h \circ e = s \circ r \circ h \circ s \circ r = s \circ r = e.$$

The proposed reconstruction of κ and δ from r and s can be seen as follows:

$$\begin{aligned} r \circ i \circ \mathbf{F}s &= \kappa \circ \mathbf{F}r \circ \mathbf{F}s = \kappa, \\ \mathbf{F}r \circ i^{-1} \circ s &= \mathbf{F}r \circ \mathbf{F}s \circ \delta = \delta. \end{aligned}$$

(ii): We have that

(a) $r \circ i = r \circ i \circ \mathbf{F}(s \circ r)$,
(b) $i^{-1} \circ s = \mathbf{F}(s \circ r) \circ i^{-1} \circ s$,
(c) $s \circ r \circ i \circ \mathbf{F}h \circ i^{-1} \circ s \circ r = h$ iff $h = e$,

and hence that

$$\begin{aligned} \kappa \circ \delta &= r \circ i \circ \mathbf{F}s \circ \mathbf{F}r \circ i^{-1} \circ s \\ &= r \circ i \circ \mathbf{F}(s \circ r) \circ i^{-1} \circ s \\ &= r \circ i \circ i^{-1} \circ s = r \circ s = \mathrm{id}_X. \end{aligned}$$

Let $f : X \to X$ with $f = \kappa \circ \mathbf{F}f \circ \delta$. As

$$\kappa \circ \mathbf{F}f \circ \delta = r \circ i \circ \mathbf{F}(s \circ f \circ r) \circ i^{-1} \circ s,$$

for $h \stackrel{\text{def}}{=} s \circ f \circ r$ we get

$$h = s \circ r \circ i \circ \mathbf{F}h \circ i^{-1} \circ s \circ r,$$

from which we get by (c) that $h = e$. But then

$$f = r \circ s \circ f \circ s \circ r = r \circ h \circ s = r \circ e \circ s = \mathrm{id}_X$$

as desired. Finally, $r : i \to \kappa$ and $s : \delta \to i^{-1}$ because

$$\kappa \circ \mathbf{F}r = r \circ i \circ \mathbf{F}s \circ \mathbf{F}r = r \circ i \qquad \text{by (a)},$$

$$\mathbf{F}s \circ \delta = \mathbf{F}s \circ \mathbf{F}r \circ i^{-1} \circ s = i^{-1} \circ s \qquad \text{by (b)}. \qquad \square$$

5 A generalized domain equation for the partial unit interval

In this section we investigate the presentation of the unit interval as an inductive retract which was introduced in Section 4.2. We first relate the presentation as an inductive retract to the presentation as a binary system developed in Section 2. We then give some examples of recursive definitions based on the inductive retraction. After that we show how the presentation as an inductive retract is simultaneously related to binary expansions and Dedekind cuts in a natural way. Finally, we briefly consider coinduction on the partial unit interval.

Recall that a functor $\mathbf{T} : \mathbf{SDom} \to \mathbf{SDom}$ was defined in Section 4.2 by

$$\mathbf{T}D = \mathcal{B} \times D \times D, \qquad \mathbf{T}f = \mathrm{id}_{\mathcal{B}} \times f \times f,$$

and that an algebra $\mathrm{constr} : \mathbf{T}\mathcal{I} \to \mathcal{I}$ and a coalgebra $\mathrm{destr} : \mathcal{I} \to \mathbf{T}\mathcal{I}$, defined by

$$\mathrm{constr}(p, y, z) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}_L(y)\ \mathrm{else}\ \mathrm{succ}_R(z),$$
$$\mathrm{destr}(x) = \langle \mathrm{left}(x), \mathrm{pred}_L(x), \mathrm{pred}_R(x) \rangle,$$

were shown to constitute an inductive retraction.

5.1 Binary algebras

We begin by relating the algebra $\mathrm{constr} : \mathbf{T}\mathcal{I} \to \mathcal{I}$ to the binary system $(\mathcal{I}, \mathrm{succ}_L, \mathrm{succ}_R)$ investigated in Section 2.

Define a *binary algebra* to be an algebra $a : \mathbf{T}D \to D$ of the form

$$\mathrm{pif} \circ (\mathrm{id}_{\mathcal{B}} \times a_L \times a_R)$$

for $a_L, a_R : D \to D$. Such maps are necessarily unique because they have to satisfy the equations $a_L(x) = a(\texttt{true}, x, y)$ and $a_R(y) = a(\texttt{false}, x, y)$. Of course, the main example of a binary algebra is constr.

Compare the following proposition to Lemma 9 and Definition 10:

Proposition 23 *Let $a : \mathbf{T}D \to D$ be a binary algebra. Then a strict continuous map $f : \mathcal{I} \to D$ is a homomorphism from* constr *to a iff*

$$\begin{aligned} f(\mathrm{succ}_L(x)) &= a_L(f(x)) \\ f(\mathrm{succ}_R(y)) &= a_R(f(y)) \\ f(\mathrm{succ}_R(x) \sqcap \mathrm{succ}_R(y)) &= a_L(f(x)) \sqcap a_R(f(y)). \end{aligned}$$

Compare the following proposition to Lemmas 8 and 9:

Proposition 24 *If there is a homomorphism from* constr *to a binary algebra $a : \mathbf{T}D \to D$ then it is the least continuous map $f : \mathcal{I} \to D$ such that*

$$f(x) = \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ a_L(f(\mathrm{pred}_L(x))\ \mathrm{else}\ a_R(f(\mathrm{pred}_R(x)).$$

Proof. By Proposition 20 we know that if there is a homomorphism constr $\to a$, then it is the least continuous function f such that

$$f = a \circ \mathbf{T}f \circ \mathrm{destr},$$

which is equivalent to the above equation. □

5.2 Examples of recursive definitions

Proposition 25 *The complement map* $\mathrm{compl} : \mathcal{I} \to \mathcal{I}$ *defined by* $\mathrm{compl}(x) = 1 - x$ *can be recursively defined by*

$$\begin{aligned} \mathrm{compl}(x) = \mathrm{pif}\ & \mathrm{left}(x) \\ \mathrm{then}\ & \mathrm{succ}_R(\mathrm{compl}(\mathrm{pred}_L(x))) \\ \mathrm{else}\ & \mathrm{succ}_L(\mathrm{compl}(\mathrm{pred}_R(x))). \end{aligned}$$

Proof. This follows from Proposition 24, because compl is easily seen to be an algebra homomorphism from constr to $\mathrm{pif} \circ (\mathrm{id}_{\mathcal{B}} \times \mathrm{succ}_R \times \mathrm{succ}_L)$. □

Proposition 26 *The map* $\exp : \mathbf{I}[0,1] \to \mathbf{I}[1,2]$ *defined by* $\exp(x) = 2^x$ *can be recursively defined by*

$$\exp(x) = \text{pif left}(x) \text{ then } \sqrt{\exp(\text{pred}_L(x))} \text{ else } \sqrt{2\exp(\text{pred}_L(x))}.$$

Proof. Define $a_L, a_R : \mathbf{I}[1,2] \to \mathbf{I}[1,2]$ by $a_L(x) = \sqrt{x}$ and $a_R(x) = \sqrt{2x}$. Then exp is again easily seen to be an algebra homomorphism from constr to $\text{pif} \circ (\text{id}_{\mathcal{B}} \times a_L \times a_R)$, and the result again follows from Proposition 24. □

More examples of definitions of elementary functions by dyadic recursion can be found in [9,11], and a recursive definition of Riemann integration can be found in [4].

5.3 Bifurcated binary expansions

The canonical solution of the domain equation $D \cong \mathbf{T}D$ is the domain $\mathcal{B}\text{Tree}$ of infinite binary trees with nodes labeled by truth values, ordered nodewise, together with the bifree algebra

$$\text{mktree} : \mathbf{T}(\mathcal{B}\text{Tree}) \to \mathcal{B}\text{Tree}$$

that maps a list $\langle p, s, t\rangle$ to the tree with root labeled by the truth value p and with left and right subtrees s and t respectively [24]:

$$\text{mktree}(p, s, t) = \quad \begin{array}{ccc} & p & \\ \swarrow & & \searrow \\ s & & t \end{array}$$

Let

$$\text{num} : \text{mktree} \to \text{constr} \quad : \mathcal{B}\text{Tree} \to \mathcal{I}$$
$$\text{bin} : \text{destr} \to \text{mktree}^{-1} : \mathcal{I} \to \mathcal{B}\text{Tree}$$

be the unique (co)algebra homomorphisms. By Lemma 17, $\text{num} \circ \text{bin} = \text{id}_{\mathcal{I}}$, so that $\mathcal{I}$ is a retract of $\mathcal{B}\text{Tree}$. The tree $\text{bin}(x)$ can be thought as a *bifurcated binary expansion* of the partial number x.

By unfolding the definitions one sees that num and bin are the unique continuous maps such that

$$\text{num}(\text{mktree}(p, s, t)) = \text{pif } p \text{ then } \text{succ}_L(\text{num}(s)) \text{ else } \text{succ}_R(\text{num}(t))$$

and

$$\text{bin}(x) = \text{mktree}(\text{left}(x), \text{bin}(\text{pred}_L(x)), \text{bin}(\text{pred}_R(y))).$$

We show that $\mathrm{bin}(x)$ is essentially the Dedekind section of $x \in \mathcal{I}$. Recall that the Dedekind section of a real number x is given by the pair of sets $\{q \in \mathbb{Q} | q < x\}$ and $\{q \in \mathbb{Q} | q > x\}$.

First, notice that an infinite binary tree over $\mathcal{B}$ can be defined as a function $2^* \to \mathcal{B}$, where $2 = \{0, 1\}$. Second, notice that the set 2^* of finite sequences over the set 2 is in bijection with the set of dyadic rationals

$$\mathbb{D} = \{m/2^n \in (0, 1) | m, n \in \mathbb{N}\},$$

via the unique map $\phi : 2^* \to \mathbb{D}$ such that

$$\phi(\epsilon) = 1/2, \qquad \phi(0\alpha) = \mathrm{succ}_L(\phi(\alpha)), \qquad \phi(1\alpha) = \mathrm{succ}_R(\phi(\alpha)).$$

Here ϵ is the empty sequence, α ranges over finite sequences, and succ_L and succ_R are considered as maps $\mathbb{D} \to \mathbb{D}$. It follows that an infinite binary tree over $\mathcal{B}$ can be considered as a function $\mathbb{D} \to \mathcal{B}$. Under this interpretation one has that

$$\begin{aligned} \mathrm{mktree}(p, s, t)(1/2) &= p, \\ \mathrm{mktree}(p, s, t)(\mathrm{succ}_L(d)) &= s(d), \\ \mathrm{mktree}(p, s, t)(\mathrm{succ}_R(d)) &= t(d). \end{aligned}$$

Proposition 27 (Dedekind-section representation)

For every $x \in \mathcal{I}$, the binary tree $\mathrm{bin}(x)$ *is the characteristic function of the dyadic Dedekind section of x, in the sense that for all $d \in \mathbb{D}$,*

$$\mathrm{bin}(x)(d) = \begin{cases} \mathtt{true} & \text{if } d < x, \\ \mathtt{false} & \text{if } d > x, \\ \bot & \text{if } d \uparrow x. \end{cases}$$

Proof. The set $\mathbb{D}$ satisfies the following dyadic induction principle: If $A \subseteq \mathbb{D}$ contains $1/2$ and is closed under succ_L and succ_R, then A is the whole of $\mathbb{D}$ (cf. Remark 7). We show that the equation holds for all x and d by dyadic induction on d. By definition of bin and mktree, we have that $\mathrm{bin}(x)(1/2) = \mathrm{left}(x)$, which establishes the base case. Now assume that the equation holds for all x and a given $d \in \mathbb{D}$. Then the fact that $\mathrm{bin}(x)(\mathrm{succ}_L(d)) = \mathrm{bin}(\mathrm{pred}_L(x))(d)$ establishes one half of the induction step, because $d < \mathrm{pred}_L(x) = \min(2x, 1)$ iff $d/2 < x$ iff $\mathrm{succ}_L(d) < x$, and, similarly, $d > \mathrm{pred}_L(x)$ iff $\mathrm{succ}_L(d) > x$. The other half is established in a symmetric fashion. $\square$

We now apply the theory of inductive retractions developed in Section 4 to show how to compute with partial real numbers via Dedekind sections. Some proofs are postponed to the end of the subsection.

We begin by looking for a recursive definition of the *normalization* idempotent

$$\mathcal{B}\mathrm{Tree} \xleftarrow{\text{norm}} \mathcal{B}\mathrm{Tree} = \mathcal{B}\mathrm{Tree} \xleftarrow{\text{bin}} \mathcal{I} \xleftarrow{\text{num}} \mathcal{B}\mathrm{Tree}$$

not involving the intermediate domain $\mathcal{I}$. Of course, the normalization idempotent has the Dedekind sections as its fixed points.

If we have an algebra $\mathrm{constr}' : \mathbf{T}(\mathcal{B}\mathrm{Tree}) \to \mathcal{B}\mathrm{Tree}$ such that bin is a homomorphism from constr to constr′, then $\mathrm{norm} : \mathrm{mktree} \to \mathrm{constr}'$, because $\mathrm{num} : \mathrm{mktree} \to \mathrm{constr}$, by definition, and homomorphisms compose. Therefore norm is the unique continuous map such that

$$\mathrm{norm} \circ \mathrm{mktree} = \mathrm{constr}' \circ \mathbf{T}\mathrm{norm},$$

or, equivalently, such that

$$\mathrm{norm}\,(\mathrm{mktree}(p, s, t)) = \mathrm{constr}'(p, \mathrm{norm}(s), \mathrm{norm}(t)),$$

which produces the desired recursive definition. Moreover, if constr′ is a binary algebra (Section 5.1), in the sense that we can find maps $\mathrm{succ}'_L, \mathrm{succ}'_R : \mathcal{B}\mathrm{Tree} \to \mathcal{B}\mathrm{Tree}$ such that

$$\mathrm{constr}' = \mathrm{pif} \circ (\mathrm{id}_{\mathcal{B}} \times \mathrm{succ}'_L \times \mathrm{succ}'_R),$$

then we can recursively define norm by

$$\mathrm{norm}\,(\mathrm{mktree}(p, s, t)) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}'_L(\mathrm{norm}(s))\ \mathrm{else}\ \mathrm{succ}'_R(\mathrm{norm}(t)).$$

Lemma 28 *Let* $\mathrm{succ}'_L, \mathrm{succ}'_R : \mathcal{B}\mathrm{Tree} \to \mathcal{B}\mathrm{Tree}$ *be continuous maps, and define*

$$\mathrm{constr}' = \mathrm{pif} \circ (\mathrm{id}_{\mathcal{B}} \times \mathrm{succ}'_L \times \mathrm{succ}'_R).$$

If the diagrams

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathrm{succ}_L} & \mathcal{I} \\ {\scriptstyle \mathrm{bin}}\downarrow & & \downarrow{\scriptstyle \mathrm{bin}} \\ \mathcal{B}\mathrm{Tree} & \xrightarrow[\mathrm{succ}'_L]{} & \mathcal{B}\mathrm{Tree} \end{array} \qquad \begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathrm{succ}'_R} & \mathcal{I} \\ {\scriptstyle \mathrm{bin}}\downarrow & & \downarrow{\scriptstyle \mathrm{bin}} \\ \mathcal{B}\mathrm{Tree} & \xrightarrow[\mathrm{succ}'_R]{} & \mathcal{B}\mathrm{Tree} \end{array}$$

commute then $\mathrm{bin} : \mathrm{constr} \to \mathrm{constr}'$.

In order to construct such maps succ'_L and succ'_R, we first consider the following construction:

Lemma 29 *Let* $\mathrm{left}'_0, \mathrm{left}'_1 : \mathcal{B}\mathrm{Tree} \to \mathcal{B}$ *be recursively defined by*

$$\mathrm{left}'_0\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{left}'_0(s)\ \mathrm{else}\ \texttt{false},$$
$$\mathrm{left}'_1\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ \texttt{true}\ \mathrm{else}\ \mathrm{left}'_1(t).$$

Then

$$\mathrm{left}'_0 = \mathrm{left}_0 \circ \mathrm{num} \qquad \textit{and} \qquad \mathrm{left}'_1 = \mathrm{left}_1 \circ \mathrm{num}.$$

The continuous maps of the hypothesis of Lemma 28 can be constructed as follows:

Lemma 30 *Let* $\mathrm{succ}'_L, \mathrm{succ}'_R : \mathcal{B}\mathrm{Tree} \to \mathcal{B}\mathrm{Tree}$ *be defined by*

$$\mathrm{succ}'_L(t) = \mathrm{mktree}(\mathrm{left}'_1(t), t, \mathrm{bin}(0)),$$
$$\mathrm{succ}'_R(t) = \mathrm{mktree}(\mathrm{left}'_0(t), \mathrm{bin}(1), t).$$

Then the diagrams displayed in Lemma 28 commute.

Notice that $\mathrm{bin}(0)$ and $\mathrm{bin}(1)$ are the binary trees with all nodes labeled by respectively `true` and `false`.

We have thus established

Proposition 31 *The normalization idempotent can be recursively defined by*

$$\mathrm{norm}\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}'_L(\mathrm{norm}(s))\ \mathrm{else}\ \mathrm{succ}'_R(\mathrm{norm}(t)).$$

Next we define $\mathrm{left}' : \mathcal{B}\mathrm{Tree} \to \mathcal{B}$ and $\mathrm{pred}'_L, \mathrm{pred}'_R : \mathcal{B}\mathrm{Tree} \to \mathcal{B}\mathrm{Tree}$ by

$$\mathrm{left}'\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{left}'_1(s)\ \mathrm{else}\ \mathrm{left}'_0(t),$$
$$\mathrm{pred}'_L\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ s\ \mathrm{else}\ \mathrm{bin}(1),$$
$$\mathrm{pred}'_R\,(\mathrm{mktree}(p,s,t)) = \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{bin}(0)\ \mathrm{else}\ t.$$

Proposition 32 *For each* $a \in \{L, R\}$*, the following diagrams commute:*

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathrm{succ}_a} & \mathcal{I} \\ {\scriptstyle \mathrm{bin}}\downarrow & & \uparrow{\scriptstyle \mathrm{num}} \\ \mathcal{B}\mathrm{Tree} & \xrightarrow[\mathrm{succ}'_a]{} & \mathcal{B}\mathrm{Tree} \end{array} \qquad \begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathrm{left}} & \mathcal{B} \\ {\scriptstyle \mathrm{bin}}\downarrow & & \uparrow{\scriptstyle \mathrm{id}} \\ \mathcal{B}\mathrm{Tree} & \xrightarrow[\mathrm{left}']{} & \mathcal{B} \end{array} \qquad \begin{array}{ccc} \mathcal{I} & \xrightarrow{\mathrm{pred}_a} & \mathcal{I} \\ {\scriptstyle \mathrm{bin}}\downarrow & & \uparrow{\scriptstyle \mathrm{num}} \\ \mathcal{B}\mathrm{Tree} & \xrightarrow[\mathrm{pred}'_a]{} & \mathcal{B}\mathrm{Tree} \end{array}$$

We finish this subsection with the proofs of the above claims.

Lemma 33 *The function* $\mathrm{bin} : \mathcal{I} \to \mathcal{B}\mathrm{Tree}$ *is multiplicative, in the sense that it preserves binary meets.*

Proof. The function bin is the least fixed-point of the continuous functional F defined by

$$F(f) = \mathrm{mktree} \circ \mathbf{T}f \circ \mathrm{destr}.$$

The function mktree is multiplicative because binary meets of triples are computed componentwise, and binary meets of trees are computed nodewise. If f is multiplicative so is $\mathbf{T}f$. Also $\mathrm{destr} = \langle \mathrm{left}, \mathrm{pred}_L, \mathrm{pred}_R \rangle$ is multiplicative because left, pred_R and pred_L are multiplicative, as it can be easily checked. Hence if f is multiplicative so is $F(f)$, because multiplicative functions are closed under composition. Therefore the least fixed-point of F is multiplicative, because the binary meet operation preserves directed joins. □

Proof. of Lemma 28: By Lemma 33, if the diagrams commute then

$$\begin{aligned}
\mathrm{bin} \circ \mathrm{constr}(p, y, z) &= \mathrm{bin}(\mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}_L(y)\ \mathrm{else}\ \mathrm{succ}_R(z)) \\
&= \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{bin}(\mathrm{succ}_L(y))\ \mathrm{else}\ \mathrm{bin}(\mathrm{succ}_R(z)) \\
&= \mathrm{pif}\ p\ \mathrm{then}\ \mathrm{succ}'_L(\mathrm{bin}(y))\ \mathrm{else}\ \mathrm{succ}'_R(\mathrm{bin}(z)) \\
&= \mathrm{succ}'(p, \mathrm{bin}(y), \mathrm{bin}(z)) \\
&= \mathrm{succ}' \circ \mathbf{T}\mathrm{bin}(p, y, z),
\end{aligned}$$

and hence $\mathrm{bin} : \mathrm{constr} \to \mathrm{constr}'$. □

Proof. of Lemma 29: Define

$$\begin{aligned}
f_0 &= \bot & f_{n+1}\,(\mathrm{mktree}(p, s, t)) &= \mathrm{pif}\ p\ \mathrm{then}\ f_n(s)\ \mathrm{else}\ \texttt{false}, \\
\mathrm{id}_0 &= \bot & \mathrm{id}_{n+1}\,(\mathrm{mktree}(p, s, t)) &= \mathrm{mktree}(p, \mathrm{id}_n(s), \mathrm{id}_n(t)).
\end{aligned}$$

Then left'_0 and $\mathrm{id} : \mathcal{B}\mathrm{Tree} \to \mathcal{B}\mathrm{Tree}$ are the least upper bounds of the chains f_n and id_n respectively. We show by induction on n that $\mathrm{left}_0 \circ \mathrm{num} \circ \mathrm{id}_n = f_n$. For the base case this is immediate, because left_0 and num are both strict. For the inductive step, notice that left_0 is multiplicative, and that hence it distributes over the parallel conditional. Also, notice that $\mathrm{left}_0 \circ \mathrm{succ}_L = \mathrm{left}_0$, because $x < 0$ iff $x/2 < 0$, and that $\mathrm{left}_0 \circ \mathrm{succ}_R(x) = \texttt{false}$, because $\mathrm{succ}_R(x) \geq 1/2$. Hence

$$
\begin{aligned}
&\mathrm{left}_0 \circ \mathrm{num} \circ \mathrm{id}_{n+1}\,(\mathrm{mktree}(p,s,t))\\
&= \mathrm{left}_0 \circ \mathrm{num}\,(\mathrm{mktree}(p, \mathrm{id}_n(s), \mathrm{id}_n(t)))\\
&= \mathrm{left}_0(\text{pif } p \text{ then } \mathrm{succ}_L(\mathrm{num}(\mathrm{id}_n(s))) \text{ else } \mathrm{succ}_R(\mathrm{num}(\mathrm{id}_n(s)))\\
&= \text{pif } p \text{ then } \mathrm{left}_0(\mathrm{succ}_L(\mathrm{num}(\mathrm{id}_n(s)))) \text{ else } \mathrm{left}_0(\mathrm{succ}_R(\mathrm{num}(\mathrm{id}_n(s))))\\
&= \text{pif } p \text{ then } \mathrm{left}_0(\mathrm{num}(\mathrm{id}_n(s))) \text{ else } \texttt{false}\\
&= \text{pif } p \text{ then } f_n(s) \text{ else } \texttt{false}\\
&= f_{n+1}\,(\mathrm{mktree}(p,s,t))\,,
\end{aligned}
$$

which finishes our inductive argument. Therefore

$$\mathrm{left}'_0 = \bigsqcup_n f_n = \bigsqcup \mathrm{left}_0 \circ \mathrm{num} \circ \mathrm{id}_n = \mathrm{left}_0 \circ \mathrm{num} \circ \bigsqcup \mathrm{id}_n = \mathrm{left}_0 \circ \mathrm{num}.$$

The proof for left'_1 is symmetric. □

Proof. of Lemma 30: Since $\mathrm{left}'_1 = \mathrm{left}_1 \circ \mathrm{num}$ by Lemma 29, and since $\mathrm{num} \circ \mathrm{bin}$, we conclude that $\mathrm{left}_1 = \mathrm{left}'_1 \circ \mathrm{bin}$. Since $\mathrm{left} \circ \mathrm{succ}_L(x) = (x/2 <_\perp 1/2) = (x <_\perp 1) = \mathrm{left}_1(x)$,

$$
\begin{aligned}
&\mathrm{bin} \circ \mathrm{succ}_L(x)\\
&= \mathrm{mktree}(\mathrm{left} \circ \mathrm{succ}_L(x), \mathrm{bin}(\mathrm{pred}_L(\mathrm{succ}_L(x))), \mathrm{bin}(\mathrm{pred}_R(\mathrm{succ}_L(x))))\\
&= \mathrm{mktree}(\mathrm{left}_1(x), \mathrm{bin}(x), \mathrm{bin}(0))\\
&= \mathrm{mktree}(\mathrm{left}'_1(\mathrm{bin}(x)), \mathrm{bin}(x), \mathrm{bin}(0))\\
&= \mathrm{succ}'_L \circ \mathrm{bin}(x).
\end{aligned}
$$

The proof for succ_R is symmetric. □

Proof. of Proposition 32: By Lemma 30, we have that $\mathrm{bin} \circ \mathrm{succ}_a = \mathrm{succ}'_a \circ \mathrm{bin}$. Hence

$$\mathrm{succ}_a = \mathrm{num} \circ \mathrm{succ}_a \circ \mathrm{bin},$$

because $\mathrm{num} \circ \mathrm{bin} = \mathrm{id}_{\mathcal{I}}$. For left we have that

$$
\begin{aligned}
&\mathrm{left}' \circ \mathrm{bin}(x)\\
&= \mathrm{left}'(\mathrm{mktree}(\mathrm{left}(x), \mathrm{bin}(\mathrm{pred}_L(x)), \mathrm{bin}(\mathrm{pred}_R(x))))\\
&= \text{pif } \mathrm{left}(x) \text{ then } \mathrm{left}'_1(\mathrm{bin}(\mathrm{pred}_L(x))) \text{ else } \mathrm{left}'_0(\mathrm{bin}(\mathrm{pred}_R(x)))\\
&= \text{pif } \mathrm{left}(x) \text{ then } \mathrm{left}_1(\mathrm{pred}_L(x)) \text{ else } \mathrm{left}_0(\mathrm{pred}_R(x)),
\end{aligned}
$$

by virtue of Lemma 29. If $\mathrm{left}(x) = \texttt{true}$ then $x < 1/2$ and hence $\mathrm{pred}_L(x) < 1$. Thus, in this case the last term is

$$\mathrm{left}_1(\mathrm{pred}_L(x)) = \texttt{true} = \mathrm{left}(x).$$

Similarly, if $\mathrm{left}(x) = \mathtt{false}$ then

$$\mathrm{left}_1(\mathrm{pred}_R(x)) = \mathtt{false} = \mathrm{left}(x).$$

Otherwise $\mathrm{left}(x) = \bot$. Then $x \sqsubseteq 1/2$ and hence $\mathrm{pred}_L(x) \sqsubseteq 1$ and $\mathrm{pred}_R(x) \sqsubseteq 0$. Therefore in this case the last term is

$$\mathrm{left}_1(\mathrm{pred}_L(x)) \sqcap \mathrm{left}_0(\mathrm{pred}_R(x)) = \bot \sqcap \bot = \bot = \mathrm{left}(x).$$

For pred_L we have that

$$\begin{aligned}
&\mathrm{num} \circ \mathrm{pred}'_L \circ \mathrm{bin}(x) \\
&= \mathrm{num} \circ \mathrm{pred}'_L(\mathrm{mktree}(\mathrm{left}(x), \mathrm{bin}(\mathrm{pred}_L(x)), \mathrm{bin}(\mathrm{pred}_R(x)))) \\
&= \mathrm{num}(\mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ \mathrm{bin}(\mathrm{pred}_L(x))\ \mathrm{else}\ \mathrm{bin}(1)) \\
&= \mathrm{num} \circ \mathrm{bin}(\mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ \mathrm{pred}_L(x)\ \mathrm{else}\ 1) \\
&= \mathrm{pif}\ \mathrm{left}(x)\ \mathrm{then}\ \mathrm{pred}_L(x)\ \mathrm{else}\ 1
\end{aligned}$$

If $\mathrm{left}(x) = \mathtt{true}$ then the last term is $\mathrm{pred}_L(x)$. If $\mathrm{left}(x) = \mathtt{false}$ then the last term is $1 = \mathrm{pred}_L(x)$. Otherwise, $\mathrm{left}(x) = \bot$. Then $x \sqsubseteq 1/2$ and $\mathrm{pred}_L(x) \sqsubseteq 1$. Hence in this case the last term is

$$\mathrm{pred}_L(x) \sqcap 1 = \mathrm{pred}_L(x).$$

For pred_R we have a symmetric proof. □

5.6 Coinduction

Dana Scott suggested that we should also consider a characterization of the partial unit interval via "co-Peano axioms" based on coinduction and coiteration. Although we don't have such a characterization yet, a coinduction principle related to the ideas of Smyth [32] and Fiore [13] immediately follows by considering the bifree **T**-algebra.

A *bisimulation* on the partial unit interval is a binary relation $\sim \subseteq \mathcal{I} \times \mathcal{I}$ such that

$$x \sim y \text{ implies that } \mathrm{left}(x) = \mathrm{left}(y) \text{ and } \mathrm{pred}_a(x) \sim \mathrm{pred}_a(y) \text{ for } a \in \{R, L\}.$$

We say that x and y are *bisimilar* if they are related by some bisimulation.

Proposition 34 (Coinduction) *If $x, y \in \mathcal{I}$ are bisimilar then $x = y$.*

Proof. Let x and y be bisimilar partial numbers. Then $\mathrm{bin}(x)$ and $\mathrm{bin}(y)$ are bisimilar trees. Hence $\mathrm{bin}(x) = \mathrm{bin}(y)$ by [13]. Therefore $x = y$ because bin is split mono. □

Of course, we can replace equalities by inequalities thus obtaining the notion of a *simulation* and a more general coinduction principle for establishing inequalities.

6 Applications to the programming language Real PCF

Real PCF [9] is an extension of the programming language PCF [29,22] with data types for the partial unit interval and the partial real line. For simplicity and without essential loss of generality, we only discuss the partial unit interval. A sketch of the treatment of the whole partial real line can be found in [11].

In this section we consider two fundamental questions: (1) What is an appropriate notion of computability for the partial real line? (2) Having found such a notion, is Real PCF universal, in the sense that all computable functions are expressible in the language?

We answer these questions by means of the recursion techniques introduced in the previous section.

6.1 The programming language PCF

PCF stands for *Programming language for Computable Functions*. It consists of the terms of Scott's LCF (*Logic of Computable Functions*). This logic was introduced in a widely circulated manuscript produced in 1969, recently published as [29], which contains the first steps towards domain theory—see also [17].

PCF is not intended as a practical programming language. Rather, it is intended as a paradigmatic programming language for the investigation of theoretical issues such as operational and denotational semantics, computational adequacy, full abstraction, program reasoning, universality [22,18]. As such, it is very concise.

We shall be deliberately informal concerning syntax and semantics. Only Scott's model of PCF will be considered, and therefore it will not be necessary to say what a model is.

PCF is a functional programming language [3]. The PCF basic data types are the flat domains $\mathcal{N}$ and $\mathcal{B}$ of natural numbers and truth values. The remaining data types are obtained by iterating the function space construction

(Section 1.3). One thus has types $[\mathcal{N} \to \mathcal{B}]$, $[[\mathcal{N} \to \mathcal{B}] \to \mathcal{B}]$, $[\mathcal{N} \to [\mathcal{N} \to \mathcal{N}]]$ and so on.

The primitive operations for the truth values type are `true`, `false` and the conditional form if p then x else y.

The primitive operations for the natural numbers type are 0, succ, pred and a test for equality with zero.

Since PCF doesn't have cartesian products, functions of the form, say, $f : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ have to be represented as "curried" functions of the form $f : \mathcal{N} \to [\mathcal{N} \to \mathcal{N}]$. In this case, instead of $f(x, y)$, one writes $f(x)(y)$.

For every type D there is a fixed point operator fix : $[D \to D] \to D$. This allows one to have recursive definitions.

Finally, one has function application and function definition. Function application produces $f(x) \in E$ from $f \in [D \to E]$ and $x \in D$. Function definition is achieved by λ-abstraction. Instead of explicitly defining a function, say, $\neg : \mathcal{B} \to \mathcal{B}$ by

$$\neg p = \text{if } p \text{ then } \texttt{false} \text{ else } \texttt{true},$$

one defines a nameless function

$$\lambda p.\text{if } p \text{ then } \texttt{false} \text{ else } \texttt{true}.$$

In practice, however, it will not be necessary to explicitly use curried functions, the fixed point operator and λ-abstraction, provided it is clear that our definitions can be easily converted to PCF notation.

All functions definable in PCF are continuous. The same is true for the extensions of PCF considered below.

6.2 Recursive types

PCF extended with recursive types is called FPC (fixed-point calculus). It was introduced by Plotkin [24]—see also [18].

First, one has more type constructors such as cartesian products and sums. Together with cartesian products come the projections $D \times E \to D$ and $D \times E \to E$, which are available as primitive. Similarly for sums.

Second, one can define types by recursion. But instead of writing, say,

$$D \cong \mathcal{B} \times D \times D,$$

one writes

$$\mu D.\, \mathcal{B} \times D \times D$$

to denote the canonical solution. But, as above, it is not necessary to explicitly use this notation provided it is clear that we know how to convert our definitions to FPC notation.

The primitive operations associated to a recursive type $D \cong \mathbf{F}D$ are the bifree algebra $i : \mathbf{F}D \to D$ and its inverse $i^{-1} : D \to \mathbf{F}D$.

6.3 The programming language Real PCF

Real PCF is PCF extended with ground data types $\mathcal{I}$ and $\mathcal{R}$—see [9]. We shall discuss only the extension with $\mathcal{I}$.

The primitive operations are the parallel conditional, the constructors succ_L, succ_R, and the destructors left, pred_L, pred_R. Actually, there are more primitive operations. But the additional ones are only needed to define the operational semantics of Real PCF. As we shall see, the ones that we have singled out are enough for our purposes.

Virtually all recursive definitions considered in the previous sections immediately give rise to recursive Real PCF programs, after appropriate conversion to fixed-point operator and λ-abstraction notation.

The ones which consider the type of boolean binary trees discussed in Section 5.3 require a further extension of Real PCF with recursive types, which could be referred to as Real FPC. Alternatively, one could encode the type of boolean binary trees in the function type $[\mathcal{N} \to \mathcal{B}]$, as it is done in [10].

6.4 Universal programming languages

Definition 35 *A programming language $\mathcal{L}$ is* universal *if every computable element of the universe of discourse of $\mathcal{L}$ is definable in $\mathcal{L}$.* □

This depends on a notion of computability in the universe of discourse. In domain theory this is achieved via the notion of an effective presentation (Section 1.5).

Before tackling Real PCF, we recall some basic facts about PCF proved by Plotkin [22]. It is easy too see that all partial recursive functions $\mathbb{N}^k \to \mathbb{N}$ are PCF definable via the natural numbers type $\mathcal{N}$. However, simple computable

functions such as the parallel conditional and the existential quantification function $\exists : [\mathcal{N} \to \mathcal{B}] \to \mathcal{B}$ defined by

$$\exists(p) = \begin{cases} \texttt{true} & \text{if } p(n) = \texttt{true} \text{ for some } n \in \mathbb{N}, \\ \texttt{false} & \text{if } p(\bot) = \texttt{false}, \\ \bot & \text{otherwise} \end{cases}$$

fail to be PCF definable. Plotkin showed that if we extend PCF with the parallel conditional then all computable first-order functions become definable, and that if we further extend PCF with the existential quantifier then all computable functions of all orders become definable.

Streicher [36] generalized this result to an extension of PCF with recursive types, parallel-or and the existential quantifier [24], and Escardó [10] generalized it to Real PCF. Here we consider Real PCF extended with recursive types.

It is straightforward to show that there exists an effective presentation of $\mathcal{I}$ that makes the primitive constructors and destructors computable. For example, any standard enumeration of the rational basis gives such an effective presentation. But one may wonder if a cleverer choice of an effective presentation would change the induced set of computable elements and functions, and this is indeed the case in general [19]. We show in Section 6.5 below that this is *not* the case in our situation.

6.5 A characterization of computability over the partial real line

Definition 36 *Two effective presentations b and b' of a domain D are* ***equivalent*** *if they can be reduced to each other, in the sense that the identity $\mathrm{id}_D : D \to D$ is computable both as a map $(D, b) \to (D, b')$ and as map $(D, b') \to (D, b)$.* □

(Notice that this is the notion of equivalence of objects in concrete categories discussed in [2], specialized to the category of effectively given domains and computable maps considered as concrete over the category of domains and continuous functions, via the forgetful functor which forgets effective presentations.)

Theorem 37 *Any two effective presentations of $\mathcal{I}$ which make* constr $: \mathbf{T}\mathcal{I} \to \mathcal{I}$ *and* destr $: \mathcal{I} \to \mathbf{T}\mathcal{I}$ *computable are equivalent.*

Proof. Let b' and b'' be two such effective presentations, and let $\mathcal{I}'$ and $\mathcal{I}''$ denote $\mathcal{I}$ endowed with b' and b''. By Corollary 19, $\mathrm{id}_{\mathcal{I}} : \mathcal{I} \to \mathcal{I}$ is the unique

algebra homomorphism constr $\to$ constr, and by Proposition 20, $\mathrm{id}_{\mathcal{I}}$ is the least fixed point of the functional

$$F : [\mathcal{I} \to \mathcal{I}] \to [\mathcal{I} \to \mathcal{I}]$$

defined by

$$F(f) = \mathrm{constr} \circ \mathbf{T}f \circ \mathrm{destr}.$$

By hypothesis, constr is computable both as a map $\mathbf{T}\mathcal{I}' \to \mathcal{I}'$ $(1')$ and as a map $\mathbf{T}\mathcal{I}'' \to \mathcal{I}''$ $(1'')$, and destr is computable both as a map $\mathcal{I}' \to \mathbf{T}\mathcal{I}'$ $(2')$ and as a map $\mathbf{T}\mathcal{I}'' \to \mathcal{I}''$ $(2'')$. By $(1'')$ and $(2')$ we conclude that F is computable as a map $[\mathcal{I}' \to \mathcal{I}''] \to [\mathcal{I}' \to \mathcal{I}'']$, which shows that $\mathrm{id}_{\mathcal{I}}$ is computable as a map $\mathcal{I}' \to \mathcal{I}''$. Similarly, by $(1')$ and $(2'')$ we conclude that it is also computable as a map $\mathcal{I}'' \to \mathcal{I}'$. $\square$

6.6 Universality of Real PCF

We prove that Real PCF extended with recursive types and the existential quantifier is universal by means of the technique introduced in [36].

Here are the main steps of the technique:

(i) Take a universal domain $\mathcal{U}$ of PCF, for example $[\mathcal{N} \to \mathcal{B}]$ (see [23]).
(ii) Show that for every domain D in the extended language there is a definable retraction

$$D \underset{s_D}{\overset{r_D}{\leftrightarrows}} \mathcal{U}$$

with $r_D \circ s_D = \mathrm{id}_D$.
(iii) Given $d \in D$ computable, $s_D(d) \in \mathcal{U}$ is computable because s_D is computable.
(iv) Since PCF extended with parallel-or and $\exists$ is universal and $\mathcal{U}$ is a PCF domain, $s_D(d)$ is definable.
(v) Hence d is definable as $d = r_D(s_D(d))$, and r_D and $s_D(d)$ are definable.
(vi) Therefore every computable element is definable.

The crucial step consists in showing that D is a definable retract of $\mathcal{U}$, and this is not so simple in the presence of recursive types. But by the general results of [36], it suffices to show that every ground type is a definable retract of $\mathcal{U}$. This has been done for the PCF ground types, so that we only need to do it for our new ground type $\mathcal{I}$.

Theorem 38 *Real PCF extended with recursive types and* $\exists : [\mathcal{N} \to \mathcal{B}] \to \mathcal{B}]$ *is a universal programming language.*

Notice that Real PCF includes a parallel conditional.

Proof. By Section 5.3, we know that num : mktree $\to$ constr and bin : destr $\to$ mktree^{-1} form a retraction with num $\circ$ bin $=$ id$_{\mathcal{I}}$. Since $\mathcal{B}$Tree is a recursive type, and since num and bin are recursively definable from constr, destr, mktree and mktree^{-1} by Proposition 20, we see that $\mathcal{I}$ is a definable retract of $\mathcal{B}$Tree. But we already know that $\mathcal{B}$Tree is a definable retract of $\mathcal{U}$. Since definable retracts compose, $\mathcal{I}$ is a definable retract of $\mathcal{U}$. $\square$

This general result does not tell the full story about definability of computable first order functions (over the partial unit interval). By means of a more direct method of proof similar to that of [22], in [11] it is shown that the existential quantifier is not needed to obtain the definability result at first-order types.

Conclusions

We have defined and studied inductive retractions. We have shown that they correspond to well-behaved quotients of bifree algebras, referred to as biquotients. We have applied this notion exclusively to the study of the partial real line and its recursion schemes and induction principles.

It might be worthwhile to look at other applications of this notion. Typically one would like to have a characterization of those (in)equational theories $\mathcal{E}$ over a signature Σ [1] such that the quotient of $T_\infty(\Sigma)$ by $\mathcal{E}$ is a biquotient. This might be interesting especially for the case of stream domains extending the work on partially commutative monoids in trace theory towards infinite behaviours.

Acknowledgements

The first named author was supported by the Brazilian agency CNPq, an EPSRC project "Programming Languages for Real Number Computation: Theory and Implementation", and an ARC project "A Computational Approach to Measure and Integration Theory".

References

[1] 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.

[2] J. Adamek, H. Herrlich, and G.E. Strecker. *Abstract and Concrete Categories*. John Wiley & Sons Inc., 1990.

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

[4] A. Edalat and M.H. Escardó. Integration in Real PCF (extended abstract). In *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 382–393, New Brunswick, New Jersey, USA, July 1996.

[5] A. Edalat and Ph. Sünderhauf. A domain-theoretic approach to computability on the real line. *This volume*.

[6] H. Egli and R.L. Constable. Computability concepts for programming languages. *Theoretical Computer Science*, 2:133–145, 1976.

[7] M.H. Escardó. Properly injective spaces and function spaces. *Topology and its Applications*. To appear.

[8] 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, Mller Centre, Cambridge, 11–14 September 1994. IC Press. 1995.

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

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

[11] M.H. Escardó. PCF extended with real numbers: A domain-theoretic approach to higher-order exact real number computation. Technical Report ECS-LFCS-97-374, Department of Computer Science, University of Edinburgh, 1997. PhD thesis, Imperial College of the University of London, 1997.

[12] M.H. Escardó and T. Streicher. Induction and recursion on the partial real line via biquotients of bifree algebras (extended abstract). In *Proceedings of the Twelveth Annual IEEE Symposium on Logic in Computer Science*, Warsaw, Polland, June 1997.

[13] M.P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Information and Computation*, 127(2):186–198, 1996.

[14] P. J. Freyd. Algebraically complete categories. *Lecture Notes in Mathematics*, 1488:95–104, 1991.

[15] P. J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the LMS Symposium, Durham, 1991.* Cambridge University Press, 1992. LMS Lecture Notes Series, 177.

[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] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF.* Springer-Verlag LNCS 78, 1979.

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

[19] A. Kanda and D. Park. When are two effectively given domains identical? In K. Weihrauch, editor, *Theoretical Computer Science* 4^{th} *GI Conference*, LNCS, 1979.

[20] D.J. Lehmann and M.B. Smyth. Algebraic specification of data types: a synthetic approach. *Math. Syst. Theory*, 14:97–139, 1981.

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

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

[23] G. Plotkin. $\mathcal{B}^\omega$ as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[24] G. Plotkin. Domains. Post-graduate Lectures in advanced domain theory, University of Edinburgh, Department of Computer Science. http:// ida.dcs.qmw.ac.uk/ sites/ other/ domain.notes.other, 1980.

[25] A. Poigné. Basic category theory. In S. Abramsky, D.M. Gabbay, and T.S.E Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 413–640. Clarendon Press, Oxford, 1992.

[26] H.L. Royden. *Real Analysis.* Collier Macmillan Publishing, third edition, 1988.

[27] D. S. Scott. Continuous lattices. In F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lectures Notes in Mathematics*, pages 97–136. Springer-Verlag, 1972.

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

[29] D. S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1993. Reprint of a manuscript produced in 1969.

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

[31] 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.

[32] M.B. Smyth. I-categories and duality. In M.P. Fourman, P.T. Johnstone, and Pitts A.M., editors, *Applications of Categories in Computer Science*, pages 270–287, Cambridge, 1992. Cambridge University Press. London Mathematical Society Lecture Notes Series 177.

[33] 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.

[34] M.B. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–783, 1982.

[35] R. Stoll. *Set Theory and Logic*. W.H. Freeman and Company, San Fransisco, 1966.

[36] T. Streicher. A universality theorem for PCF with recursive types, parallel-or and $\exists$. *Mathematical Structures for Computing Science*, 4(1):111 – 115, 1994.