Operational domain theory and topology of a sequential programming language

Martín Escardó Ho Weng Kin
School of Computer Science, University of Birmingham, UK, revised 17th October 2005

Abstract

A number of authors have exported domain-theoretic techniques from denotational semantics to the operational study of contextual equivalence and preorder. We further develop this, and, moreover, we additionally export topological *techniques. In particular, we work with an operational notion of compact set and show that total programs with values on certain types are uniformly continuous on compact sets of total elements. We apply this and other conclusions to prove the correctness of non-trivial programs that manipulate infinite data. What is interesting is that the development applies to* sequential *programming languages.*

1. Introduction

Domain theory and topology in programming language semantics have been applied to manufacture and study *denotational* models, e.g. the Scott model of PCF [23]. As is well known, for a sequential language like this, the match of the model with the operational semantics is imprecise: computational adequacy holds but full abstraction fails [21].

The main achievement of the present work is a reconciliation of a good deal of domain theory and topology with sequential computation. This is accomplished by sidestepping denotational semantics and reformulating domain-theoretic and topological notions directly in terms of programming concepts, interpreted in an operational way.

Regarding domain theory, we replace directed sets by rational chains, which we observe to be equivalent to programs defined on a "vertical natural numbers" type. Many of the classical definitions and theorems go through with this modification. In particular, (1) rational chains have suprema in the contextual order, (2) programs of functional type preserve suprema of rational chains, (3) every element (closed term) of any type is the supremum of a rational chain of finite elements, (4) two programs of functional type are contextually equivalent iff they produce a contextually equivalent result for every finite input. Moreover, we have an SFP-style characterization of finiteness using rational chains of deflations, a Kleene-Kreisel density theorem for total elements, and a number of continuity principles based on finite elements.

Regarding topology, we define open sets of elements via programs with values on a "Sierpinski" type, and compact sets of elements via Sierpinski-valued universal-quantification programs. Then (1) the open sets of any type are closed under the formation of finite intersections and rational unions, (2) open sets are "rationally Scott open", (3) compact sets satisfy the "rational Heine–Borel property", (4) total programs with values on certain types are uniformly continuous on compact sets of total elements.

In order to be able to formulate certain specifications of higher-type programs without invoking a denotational semantics, we work with a "data language" for our programming language, which consists of the latter extended with first-order "oracles". The idea is to have a more powerful environment in order to get stronger program specifications. We observe that program equivalence defined by ground data contexts coincides with program equivalence defined by ground program contexts, but the notion of totality changes. (It is worth mentioning that the resulting data language for PCF defines precisely the elements of games models [1, 12], with the programming language capturing the effective parts of the models; similarly, the resulting data language for PCF extended with parallel-or and Plotkin's existential quantifier defines precisely the elements of the Scott model, again with the programming language capturing the effective part [21, 8].)

We illustrate the scope and flexibility of the theory by applying our conclusions to prove the correctness of non-trivial programs that manipulate infinite data. We take one such example from [24]. In order to avoid having exact real-number computation as a prerequisite, as in that reference, we consider modified versions of the program and its specification that retain their essential aspects. We show that the given specification and proof in the Scott model can be directly understood in our operational setting.

Although our development is operational, we never invoke evaluation mechanisms directly. We instead rely on known extensionality, monotonicity, and rational-chain principles for contextual equivalence and order. Moreover, with the exception of the proof of the density theorem, we don't perform syntactic manipulations with terms.

Related work. The idea that order-theoretic techniques from domain theory can be directly understood in terms of operational semantics goes back to Mason, Smith, Talcot [15] and Sands (see Pitts [19] for references). Already in [15], one can find, in addition to rational-chain principles, two equivalent formulations of an operational notion of finiteness. One is analogous to our Definition 4.1 but uses directed sets of closed terms rather than rational chains, and the other is analogous to our Theorem 4.8. In addition to redeveloping their formulations in terms of rational chains, here we add a topological characterization (Theorem 4.15).

The idea that topological techniques can be directly understood in terms of operational semantics, and, moreover, are applicable to sequential languages, is due to Escardó [8]. In particular, we have taken our operational notion of compactness and some material about it from that reference. The main novelty here is a uniform-continuity principle, which plays a crucial role in the sample applications given in Section 7. This is inspired by unpublished work by Andrej Bauer and Escardó on synthetic analysis in (sheaf and realizability) toposes.

The idea of invoking a data language to formulate higher-type program specifications in a sequential operational setting is already developed in [8] and is related to relative realizability [4] and TTE [27].

2. Preliminaries

The programming language. We work with a simply-typed λ-calculus with function and finite-product types, fixed-point recursion, and base types `Nat` for natural numbers and `Bool` for booleans. We regard this as a programming language under the call-by-name evaluation strategy. In summary, we work with PCF extended with finite-product types (see e.g. one of the references [10, 19]). Other possibilities are briefly discussed in Section 8.

For clarity of exposition, we also include a *Sierpinski* base type Σ and a *vertical-natural-numbers* base type $\overline{\omega}$, although such types can be easily encoded in other existing types if one so desires (e.g. via retractions). We have the following term-formation rules for them:

(1) $\top\colon \Sigma$ is a term.
(2) If $M\colon \Sigma$ and $N\colon \sigma$ are terms then $(\text{if } M \text{ then } N)\colon \sigma$ is a term.
(3) If $M\colon \overline{\omega}$ is a term then $(M+1)\colon \overline{\omega}$, $(M-1)\colon \overline{\omega}$, and $(M > 0)\colon \Sigma$ are terms.

The only value (or canonical form) of type Σ is $\top$, and the values of type $\overline{\omega}$ are the terms of the form $M+1$. The role of zero is played by divergent computations, and a term $(M > 0)$ can be thought of as a convergence test. The big-step operational semantics for these constructs is given by the following evaluation rules:

(i) If $M \Downarrow \top$ and $N \Downarrow V$ then $(\text{if } M \text{ then } N) \Downarrow V$.
(ii) If $M \Downarrow N+1$ and $N \Downarrow V$ then $M-1 \Downarrow V$.
(iii) If $M \Downarrow M'+1$ then $M > 0 \Downarrow \top$.

For any type σ, we define $\perp_\sigma = \mathrm{fix}\, x.x$, where fix denotes the fixed-point recursion construct. In what follows, if $f\colon \sigma \to \sigma$ is a closed term, we shall write $\mathrm{fix}\, f$ as an abbreviation for $\mathrm{fix}\, x.f(x)$.

Oracles. We also consider the extension of the programming language with the following term-formation rule:

(4) If $\Omega\colon \mathbb{N} \to \mathbb{N}$ is any function, computable or not, and $N\colon$ `Nat` is a term, then $\Omega N\colon$ `Nat` is a term.

Then the operational semantics is extended by the rule:

(iv) If $N \Downarrow n$ and $\Omega(n) = m$ then $\Omega N \Downarrow m$.

We think of Ω as an external input or *oracle*, and of the equation $\Omega(n) = m$ as a query with question n and answer m. Of course, the extension of the language with oracles is no longer a *programming language*. We shall regard it as a *data language* in Section 6.

Underlying language for Sections 3–5. We take it to be either (1) the programming language introduced above, (2) its extension with oracles, (3) its extension with parallel features, such as parallel-or and Plotkin's existential quantifier, or else (4) its extension with both oracles and parallel features. The conclusions of those sections hold for the four possibilities, at no additional cost. To emphasize that a closed term doesn't include oracles, we refer to it as a *program*.

Notation for contextual equivalence and (pre)order. We write $M = N$ and $M \sqsubseteq N$ to denote (ground) contextual equivalence and order of terms of the same type.

Elements of a type. By an *element* of a type we mean a closed term of that type. We adopt usual set-theoretic notation for the elements of a type in the sense just defined. For example, we write $x \in \sigma$ and $f \in (\sigma \to \tau)$ to mean that x is an element of type σ and f is an element of type $\sigma \to \tau$.

The elements of Σ***.*** The elements $\perp$ and $\top$ of Σ are contextually inequivalent and any element of Σ is equivalent to one of them. We think of Σ as a type of results of observations or semidecisions, with $\top$ as "observable true" and $\perp$ as "unobservable false".

The elements of $\overline{\omega}$***.*** We denote by ∞ the element $\mathrm{fix}\, x.x+1$ of $\overline{\omega}$, and, by an abuse of notation, for $n \in \mathbb{N}$ we write n to denote the element $\mathrm{succ}^n(\perp)$ of $\overline{\omega}$, where $\mathrm{succ}(x) = x+1$. The elements $0, 1, 2, \dots, n, \dots, \infty$ of $\overline{\omega}$ are all contextually inequivalent, and any element of $\overline{\omega}$ is contextually equivalent to one of them. Notice that the equivalences $0-1 = 0$, $(x+1)-1 = x$, $(0>0) = \perp$ and $(x+1>0) = \top$ hold for $x \in \overline{\omega}$. In particular, $\infty - 1 = \infty$ and $(\infty > 0) = \top$.

Extensionality and monotonicity. Contextual equivalence is a congruence: If $f = g$ and $x = y$ then $f(x) = g(y)$ for any $f, g \in (\sigma \to \tau)$ and $x, y \in \sigma$. Moreover, application is extensional: $f = g$ if $f(x) = g(x)$ for all $x \in \sigma$.

Regarding the contextual order, we have that application is monotone: If $f \sqsubseteq g$ and $x \sqsubseteq y$ then $f(x) \sqsubseteq g(y)$ for any $f, g \in (\sigma \to \tau)$ and $x, y \in \sigma$. Moreover, it is order-extensional: $f \sqsubseteq g$ if $f(x) \sqsubseteq g(x)$ for all $x \in \sigma$. Standard congruence, extensionality and monotonicity principles also hold for product types. Additionally, $\perp_\sigma$ is the least element of σ.

Rational chains. For any $g \in (\tau \to \tau)$ and any $h \in (\tau \to \sigma)$, the sequence $h(g^n(\perp))$ is increasing and has $h(\operatorname{fix} g)$ as a least upper bound in the contextual order:

$$h(\operatorname{fix} g) = \bigsqcup_n h(g^n(\perp)).$$

A sequence x_n of elements of a type σ is called a *rational chain* if there exist $g \in (\tau \to \tau)$ and $h \in (\tau \to \sigma)$ with $x_n = h(g^n(\perp))$.

Proofs. The facts stated in this section are all well known. The extensionality, monotonicity and rational-chain principles follow directly from Milner's construction [16]. Even though full abstraction of the Scott model fails for sequential languages, proofs exploiting computational adequacy are possible [13] (see [18]). Proofs using game semantics can be found in [1, 12], and operational proofs can be found in [19, 20] (where an earlier operational proof of the rational-chains principle is attributed to Sands). For a call-by-value untyped language, an operational proof of the rational-chains principle was previously developed in [15]. Regarding the above description of the elements of the vertical-natural-numbers type, a denotational proof using adequacy is easy, and operational proofs are obtained applying [9] or [19].

3. Rational chains and open sets

Lemma 3.1. *The sequence $0, 1, 2, \ldots, n, \ldots$ in $\overline{\omega}$ is a rational chain with least upper bound ∞, and*

$$l(\infty) = \bigsqcup_n l(n) \text{ for every } l \in (\overline{\omega} \to \sigma).$$

Proof. $n = \operatorname{succ}^n(\perp)$ and $\infty = \operatorname{fix} \operatorname{succ}$. □

Moreover, this is the "generic rational chain" with "generic lub ∞" in the following sense:

Lemma 3.2. *A sequence $x_n \in \sigma$ is a rational chain if and only if there exists $l \in (\overline{\omega} \to \sigma)$ such that for all $n \in \mathbb{N}$,*

$$x_n = l(n), \text{ and hence such that } \bigsqcup_n x_n = l(\infty).$$

Proof. ($\Rightarrow$): Given $g \in (\tau \to \tau)$ and $h \in (\tau \to \sigma)$ with $x_n = h(g^n(\perp))$, recursively define

$$f(y) = \text{if } y > 0 \text{ then } g(f(y-1)).$$

Then $f(n) = g^n(\perp)$ and hence we can take $l = h \circ f$.

($\Leftarrow$): Take $h = l$ and $g(y) = y + 1$. □

Elements of functional type are "rationally continuous" in the following sense:

Proposition 3.3. *If $f \in (\sigma \to \tau)$ and x_n is a rational chain in σ, then*

1. *$f(x_n)$ is a rational chain in τ, and*
2. *$f(\bigsqcup_n x_n) = \bigsqcup_n f(x_n)$.*

Proof. By Lemma 3.2, there is $l \in (\overline{\omega} \to \sigma)$ such that $x_n = l(n)$. Then the definition $l'(y) = f(l(y))$ and the same lemma show that $f(x_n)$ is a rational chain. By two applications of Lemma 3.1, $f(\bigsqcup_n x_n) = f(l(\infty)) = l'(\infty) = \bigsqcup_n l'(n) = \bigsqcup_n f(l(n)) = \bigsqcup_n f(x_n)$. □

Corollary 3.4. *For any rational chain f_n in $(\sigma \to \tau)$ and any $x \in \sigma$,*

1. *$f_n(x)$ is a rational chain in τ, and*
2. *$(\bigsqcup_n f_n)(x) = \bigsqcup_n f_n(x)$.*

Proof. Apply the proposition to $F \in ((\sigma \to \tau) \to \tau)$ defined by $F(f) = f(x)$. □

Definition 3.5. We say that a set U of elements of a type σ is *open* if there is $\chi_U \in (\sigma \to \Sigma)$ such that for all $x \in \sigma$,

$$\chi_U(x) = \top \iff x \in U.$$

If such an element χ_U exists then it is unique up to contextual equivalence, and we refer to it as the *characteristic function* of U.

We say that a sequence of open sets in σ is a rational chain if the corresponding sequence of characteristic functions is rational in the function type $(\sigma \to \Sigma)$. The following says that the open sets of any type form a "rational topology":

Proposition 3.6. *For any type, the open sets are closed under the formation of finite intersections and rational unions.*

Proof. Finite intersections: $\chi_{\bigcap \emptyset}(x) = \top$ and $\chi_{U \cap V}(x) = \chi_U(x) \wedge \chi_V(x)$, where $p \wedge q = \text{if } p \text{ then } q$.

Rational unions: Because $U \subseteq V$ iff $\chi_U \sqsubseteq \chi_V$, we have that if $l \in (\overline{\omega} \to (\sigma \to \Sigma))$ and $l(n)$ is the characteristic function of U_n then $l(\infty) = \bigsqcup_n \chi_{U_n} = \chi_{\bigcup_n U_n}$. □

However, unless the language has parallel features, the open sets don't form topologies in the usual sense:

Proposition 3.7. *The following are equivalent.*

1. *For every type, the open sets are closed under the formation of finite unions.*
2. *There is $(\vee) \in (\Sigma \times \Sigma \to \Sigma)$ such that*

$$p \vee q = \top \iff p = \top \text{ or } q = \top.$$

Proof. ($\Uparrow$): $\chi_{\bigcup \emptyset}(x) = \perp$ and $\chi_{U \cup V}(x) = \chi_U(x) \vee \chi_V(x)$.

($\Downarrow$): $U = \{(p, q) \mid p = \top\}$ and $V = \{(p, q) \mid q = \top\}$ are open in the type $\Sigma \times \Sigma$ because they have the first and second projections as their characteristic functions. Hence the set $U \cup V$ is also open, and so there is $\chi_{U \cup V}$ such that $\chi_{U \cup V}(p, q) = \top$ iff $(p, q) \in U \cup V$ iff $(p, q) \in U$ or $(p, q) \in V$ iff $p = \top$ or $q = \top$. Therefore $(\vee) = \chi_{U \cup V}$ gives the desired conclusion. □

Moreover, even if parallel features are included, closure under arbitrary unions fails in general (but see [8, Chapter 4]). The following easy observation says that elements of functional type are continuous in the topological sense:

Proposition 3.8. *For any $f \in (\sigma \to \tau)$ and any open subset V of τ, the set $f^{-1}(V) = \{x \in \sigma \mid f(x) \in V\}$ is open in σ.*

Proof. If $\chi_V \in (\tau \to \Sigma)$ is the characteristic function of the set V then $\chi_V \circ f \in (\sigma \to \Sigma)$ is that of $f^{-1}(V)$. □

The following says that the contextual order is the "specialization order" of the topology:

Proposition 3.9. *For $x, y \in \sigma$, the relation $x \sqsubseteq y$ holds iff $x \in U$ implies $y \in U$ for every open subset U of σ.*

Proof. Ground contexts of type Σ suffice to test the operational preorder — see e.g. [19, Remark 2.10]. Because x and y are closed terms, applicative contexts, i.e. characteristic functions of open sets, suffice. □

Open sets are "rationally Scott open":

Proposition 3.10. *For any open set U in a type σ,*

1. *if $x \in U$ and $x \sqsubseteq y$ then $y \in U$, and*
2. *if x_n is a rational chain with $\bigsqcup x_n \in U$, then there is $n \in \mathbb{N}$ such that already $x_n \in U$.*

Proof. By monotonicity and rational continuity of χ_U. □

4. Finite elements

We develop a number of equivalent formulations of a notion of finiteness. Corollary 4.3 says that an element b is finite iff any attempt to build b as the lub of a rational chain already has b as a building block. The official definition is a bit subtler:

Definition 4.1. An element b is called (rationally) *finite* if for every rational chain x_n with $b \sqsubseteq \bigsqcup_n x_n$, there is n such that already $b \sqsubseteq x_n$.

The types of our language are "rationally algebraic" in the following sense:

Theorem 4.2. *Every element of any type is the least upper bound of a rational chain of finite elements.*

A proof of this will be given later in this section. For the moment, we develop some consequences.

Corollary 4.3. *An element b is finite if and only if for every rational chain x_n with $b = \bigsqcup_n x_n$, there is n such that already $b = x_n$.*

Proof. ($\Rightarrow$): If $b = \bigsqcup_n x_n$ then $b \sqsubseteq \bigsqcup_n x_n$ and hence $b \sqsubseteq x_n$ for some n. But, by definition of upper bound, we also have $b \sqsupseteq x_n$. Hence $b = x_n$, as required.

($\Leftarrow$): By Theorem 4.2, there is a rational chain x_n of finite elements with $b = \bigsqcup_n x_n$. By the hypothesis, $b = x_n$ for some n, which shows that b is finite. □

The following provides a proof method for contextual equivalence based on finite elements:

Proposition 4.4. *$f = g$ holds in $(\sigma \to \tau)$ iff $f(b) = g(b)$ for every finite $b \in \sigma$.*

Proof. ($\Rightarrow$): Contextual equivalence is an applicative congruence. ($\Leftarrow$): By extensionality it suffices to show that $f(x) = g(x)$ for any $x \in \sigma$. By Theorem 4.2, there is a rational chain b_n of finite elements with $x = \bigsqcup_n b_n$. Hence, by two applications of rational continuity and one of the hypothesis, $f(x) = f(\bigsqcup_n b_n) = \bigsqcup_n f(b_n) = \bigsqcup_n g(b_n) = g(\bigsqcup_n b_n) = g(x)$, as required. □

Of course, the above holds with contextual equivalence replaced by contextual order. Another consequence of Theorem 4.2 is a third continuity principle:

Proposition 4.5. *For any $f \in (\sigma \to \tau)$, any $x \in \sigma$ and any finite $c \sqsubseteq f(x)$, there is a finite $b \sqsubseteq x$ with $c \sqsubseteq f(b)$.*

Proof. By Theorem 4.2, x is the lub of a rational chain b_n of finite elements. By rational continuity, $c \sqsubseteq \bigsqcup_n f(b_n)$. By finiteness of c, there is n with $c \sqsubseteq f(b_n)$. □

Corollary 4.6. *If U is open and $x \in U$, then there is a finite $b \sqsubseteq x$ such that already $b \in U$.*

Proof. The hypothesis gives $\top \sqsubseteq \chi_U(x)$, and so there is some finite $b \sqsubseteq x$ with $\top \sqsubseteq \chi_U(b)$ because $\top$ is finite. To conclude, use maximality of $\top$. □

In order to prove Theorem 4.2, we invoke the following concepts (see e.g. [2]):

Definition 4.7.

1. A *deflation* on a type σ is an element of type $(\sigma \to \sigma)$ that (*i*) is below the identity of σ, and (*ii*) has finite image modulo contextual equivalence.
2. A (rational) *SFP structure* on a type σ is a rational chain id_n of idempotent deflations with $\bigsqcup_n \mathrm{id}_n = \mathrm{id}$, the identity of σ.
3. A type is (rationally) *SFP* if it has an SFP structure.

Theorem 4.8.

1. *Each type of the language is SFP.*
2. *For any SFP structure id_n on a type σ, an element $b \in \sigma$ is finite if and only if $b = \mathrm{id}_n(b)$ for some $n \in \mathbb{N}$.*

In particular, because id_n is idempotent, $\mathrm{id}_n(x)$ is finite and hence any $x \in \sigma$ is the lub of the rational chain $\mathrm{id}_n(x)$ and therefore Theorem 4.2 follows.

Proof. (1): Lemma 4.13 below.

(2)($\Rightarrow$): The inequality $b \sqsupseteq \mathrm{id}_n(b)$ holds because id_n is a deflation. For the other inequality, we first calculate $b = (\bigsqcup_n \mathrm{id}_n)(b) = \bigsqcup_n \mathrm{id}_n(b)$ using Corollary 3.4. Then by finiteness of b, there is n with $b \sqsubseteq \mathrm{id}_n(b)$.

(2)($\Leftarrow$): To show that b is finite, let x_i be a rational chain with $b \sqsubseteq \bigsqcup_i x_i$. Then $b = \mathrm{id}_n(b) \sqsubseteq \mathrm{id}_n(\bigsqcup_i x_i) = \bigsqcup_i \mathrm{id}_n(x_i)$ by rational continuity of id_n. Because id_n has finite image, the set $\{\mathrm{id}_n(x_i) \mid i \in \mathbb{N}\}$ is finite and hence has a maximal element, which is its lub. That is, there is $i \in \mathbb{N}$ with $b \sqsubseteq \mathrm{id}_n(x_i)$. But $\mathrm{id}_n(x_i) \sqsubseteq x_i$ and hence $b \sqsubseteq x_i$, as required. $\square$

We additionally have the following proposition.

Definition 4.9. By a *finitary type* we mean a type that is obtained from Σ and `Bool` by finitely many applications of the product- and function-type constructions.

Proposition 4.10. *SFP structures $\mathrm{id}^\sigma_n \in (\sigma \to \sigma)$ can be chosen for each type σ in such a way that*

1. *id^σ_n is the identity for every finitary type σ,*
2. *$\mathrm{id}^{\sigma\to\tau}_n(f)(x) = \mathrm{id}^\tau_n(f(\mathrm{id}^\sigma_n(x)))$,*
3. *$\mathrm{id}^{\sigma\times\tau}_n(x,y) = (\mathrm{id}^\sigma_n(x), \mathrm{id}^\tau_n(y))$.*

Item 1 gives:

Corollary 4.11. *Every element of any finitary type is finite.*

The other two give the following consequence, whose proof uses the fact that for any SFP structure id_n on a type σ, if $\mathrm{id}_n(x) = x$ then $\mathrm{id}_k(x) = x$ for any $k \geq n$.

Corollary 4.12. *1. If $f \in (\sigma \to \tau)$ and $x \in \sigma$ are finite then so is $f(x) \in \tau$.*
2. If $x \in \sigma$ and $y \in \tau$ are finite then so is $(x,y) \in (\sigma\times\tau)$.

Proof. (1): If f and x are finite, then there are m and n with $f = \mathrm{id}_m(f)$ and $x = \mathrm{id}_n(x)$. Let $k = \max(m,n)$. By Proposition 4.10, $f(x) = \mathrm{id}_k(f)(\mathrm{id}_k(x)) = \mathrm{id}_k(f(x))$, which shows that $f(x)$ is finite. (2): Similar. $\square$

To prove Theorem 4.8(1) and Proposition 4.10, we construct, by induction on σ, programs

$$\mathrm{d}^\sigma : \overline{\omega} \to (\sigma \to \sigma).$$

For the base case, we define

$$\mathrm{d}^{\mathtt{Bool}}(x)(p) = p, \qquad \mathrm{d}^{\Sigma}(x)(p) = p,$$
$$\mathrm{d}^{\mathtt{Nat}}(x)(k) = \text{ if } x > 0 \text{ then } \text{if } k == 0 \text{ then } 0 \text{ else } 1 + \mathrm{d}^{\mathtt{Nat}}(x-1)(k-1),$$
$$\mathrm{d}^{\overline{\omega}}(x)(y) = \text{if } x > 0 \wedge y > 0 \text{ then } 1 + \mathrm{d}^{\overline{\omega}}(x-1)(y-1).$$

Notice that "$x > 0$" and "$x > 0 \wedge y > 0$" are terms of Sierpinski type and hence the "if" symbols that precede them don't have corresponding "else" clauses. For the induction step, we define

$$\begin{aligned} \mathrm{d}^{\sigma\to\tau}(x)(f)(y) &= \mathrm{d}^\tau(x)(f(\mathrm{d}^\sigma(x)(y))), \\ \mathrm{d}^{\sigma\times\tau}(x)(y,z) &= (\mathrm{d}^\sigma(x)(y), \mathrm{d}^\tau(x)(z)). \end{aligned}$$

Lemma 4.13. *The rational chain $\mathrm{id}^\sigma_n \stackrel{\text{def}}{=} \mathrm{d}^\sigma(n)$ is an SFP structure on σ for every type σ.*

Proof. By induction on σ. For the base case, only $\sigma = \overline{\omega}$ is non-trivial. First show by induction on n that, for every $n \in \mathbb{N}$, $\mathrm{d}^{\overline{\omega}}(n)(y) = \min(n,y)$. Hence $\mathrm{d}^{\overline{\omega}}(n)$ is idempotent and below the identity, and has image $\{0, 1, \ldots, n\}$. Now calculate, for $k \in \mathbb{N}$, $\mathrm{d}^{\overline{\omega}}(\infty)(k) = \bigsqcup_n \mathrm{d}^{\overline{\omega}}(n)(k) = \bigsqcup_n \min(n,k) = k$. Hence $\mathrm{d}^{\overline{\omega}}(\infty)(\infty) = \bigsqcup_k \mathrm{d}^{\overline{\omega}}(\infty)(k) = \bigsqcup_k k = \infty$. By extensionality, $\mathrm{d}^{\overline{\omega}}(\infty)$ is the identity. The induction step is straightforward. $\square$

This proves Theorem 4.8(1). Proposition 4.10(1) is easily established by induction on finitary types, and conditions (2) and (3) are immediate.

We now develop a topological characterization of the notion of finiteness. We say that an open set in σ has *finite characteristic* if its characteristic function is a finite element of the function type $(\sigma \to \Sigma)$.

Lemma 4.14. *For any open set U in σ and any $n \in \mathbb{N}$, let*

$$U^{(n)} \stackrel{\text{def}}{=} \mathrm{id}_n^{-1}(U) = \{x \in \sigma \mid \mathrm{id}_n(x) \in U\}.$$

1. *The open set $U^{(n)} \subseteq U$ has finite characteristic.*
2. *The set $\{U^{(n)} \mid U \text{ is open in } \sigma\}$ has finite cardinality.*
3. *U has finite characteristic iff $U = U^{(n)}$ for some n.*
4. *The chain $U^{(n)}$ is rational and $U = \bigcup_n U^{(n)}$.*

Proof. (1) and (3): $\mathrm{id}_n(\chi_U)(x) = \mathrm{id}_n(\chi_U(\mathrm{id}_n(x))) = \chi_U(\mathrm{id}_n(x))$, and hence $\mathrm{id}_n(\chi_U)$ is the characteristic function of $U^{(n)}$.

(2): Any two equivalent characteristic functions classify the same open set and $\mathrm{id}^{\sigma\to\Sigma}_n$ has finite image modulo contextual equivalence.

(4): $\mathrm{id}_n(\chi_U)$ is a rational chain with lub χ_U, i.e. $\chi_U(x) = \top$ iff $\mathrm{id}_n(\chi_U)(x) = \top$ for some n. $\square$

Theorem 4.15. *An element $b \in \sigma$ is finite if and only if the set $\uparrow b \stackrel{\text{def}}{=} \{x \in \sigma \mid b \sqsubseteq x\}$ is open.*

Proof. ($\Rightarrow$): By Proposition 3.9, for any $x \in \sigma$ we have that $\uparrow x = \bigcap\{U \mid U \text{ is open and } x \in U\}$. Because b is finite, there is n such that $\mathrm{id}_n(b) = b$. Hence if b belongs to an open set U then $b \in U^{(n)} \subseteq U$ by Lemma 4.14(1). This shows that $\uparrow b = \bigcap\{U^{(n)} \mid U \text{ is open and } b \in U\}$. But this is the intersection of a set of finite cardinality by Lemma 4.14(2) and hence open by Proposition 3.6.

($\Leftarrow$): If $b \sqsubseteq \bigsqcup_n x_n$ holds for a rational chain x_n, then $\bigsqcup_n x_n \in \uparrow b$ and hence $x_n \in \uparrow b$ for some x_n by Proposition 3.10(2), i.e. $b \sqsubseteq x_n$. $\square$

Corollary 4.16. *Every open set is a union of open sets of the form* $\uparrow b$ *with* b *finite.*

Proof. If x belongs to an open set U then $x \in \uparrow b \subseteq U$ for some finite b by Corollary 4.6 and Proposition 3.10(1). $\square$

Remark 4.17. (1) Notice that the proof of Theorem 4.15($\Rightarrow$) is not constructive. The reason is that we implicitly use the fact that a subset of a finite set is finite. In general, however, it is not possible to finitely enumerate the members of a subset of a finite set unless the defining property of the subset is decidable, and here it is only semidecidable. (2) Moreover, this non-constructivity in the theorem is unavoidable. In fact, if we had a constructive procedure for finding $\chi_{\uparrow b}$ for every finite b, then we would be able to semidecide contextual equivalence for finite elements, because $b = c$ iff $\chi_{\uparrow b}(c) = \top = \chi_{\uparrow c}(b)$. As all elements of finitary PCF are finite, and contextual equivalence is co-semidecidable for finitary PCF, this would give a decision procedure for equivalence, contradicting [14].

We now develop an operational version of the Kleene–Kreisel density theorem for total elements [7].

Definition 4.18. An element of ground type is *total* iff it is maximal. An element $f \in (\sigma \to \tau)$ is *total* iff $f(x) \in \tau$ is total whenever $x \in \sigma$ is total. An element of type $(\sigma \times \tau)$ is total iff its projections into σ and τ are total.

It is easy to see that any type has a total element. In order to cope with the fact that the only total element of $\overline{\omega}$, namely ∞, is defined by fixed-point recursion, we need:

Lemma 4.19. *If* x *is an element of any type defined from total elements* $y_1, \dots, y_n$ *in such a way that the only occurrences of the fixed-point combinator in* x *are those of* $y_1, \dots, y_n$*, if any, then* x *is total.*

Proof. Define a term with free variables to be total if every instantiation of its free variables by total elements produces a total element, and then proceed by induction on the formation of x from $y_1, \dots, y_n$. $\square$

Theorem 4.20. *Every finite element is below some total element. Hence any inhabited open set has a total element.*

Proof. For each type τ and each $n \in \mathbb{N}$, define programs $F^\tau : \overline{\omega} \to ((\tau \to \tau) \to \tau)$ and $G^\tau_n : (\tau \to \tau) \to \tau$ by

$F(x)(f) = \text{if } x > 0 \text{ then } f(F(x-1)(f)),$
$G_n(f) = f^n(t) \text{ for some chosen } t \in \tau \text{ total.}$

Then $F(\infty) = \text{fix}$, $F(n) \sqsubseteq G_n$ and G_n is total.

Now, given a finite element b of any type, choose a fresh syntactic variable x of type $\overline{\omega}$, and define a term $\tilde{b}$ from b by replacing all occurrences of fix^τ by the term $F^\tau(x)$. Then $b = (\lambda x.\tilde{b})(\infty)$. Because b is finite, there is some $n \in \mathbb{N}$ such that already $b = (\lambda x.\tilde{b})(n)$.

To conclude, construct a term $\hat{b}$ from b by replacing all occurrences of fix^τ by G^τ_n. Then $\hat{b}$ is total by Lemma 4.19, and $(\lambda x.\tilde{b})(n) \sqsubseteq \hat{b}$ and hence $b \sqsubseteq \hat{b}$ by transitivity. $\square$

We finish this section by considering a continuity principle for two special kinds of functions. For elements x and y of the same type, define

$x =_n y \iff \text{id}_n(x) = \text{id}_n(y).$

We refer to the function type $(\texttt{Nat} \to \texttt{Nat})$ as the *Baire type* and denote it by `Baire`. (Then the set of *total* elements of `Baire` is an operational manifestation of the *Baire space* of classical topology.) The following is easily proved:

Lemma 4.21. *Define* $\overline{\text{id}}_n : \texttt{Baire} \to \texttt{Baire}$ *by* $\overline{\text{id}}_n(s) = \lambda i.\text{if } i < n \text{ then } s(i) \text{ else } \bot$*. Then* $\overline{\text{id}}_n(s)$ *is finite and above* $\text{id}_n(s)$*, and if* $s, t \in \texttt{Baire}$ *are total then*

$\overline{\text{id}}_n(s) \sqsubseteq t \implies s =_n t.$

Proposition 4.22. *For total* $f \in (\sigma \to \texttt{Baire})$ *and* $x \in \sigma$*,*

$$\forall \epsilon \in \mathbb{N}\, \exists \delta \in \mathbb{N}\ \forall \textit{total } y \in \sigma,\ x =_\delta y \Rightarrow f(x) =_\epsilon f(y).$$

Proof. Because $\overline{\text{id}}_\epsilon(f(x))$ is finite and below $f(x)$, there is δ such that already $\overline{\text{id}}_\epsilon(f(x)) \sqsubseteq f(\text{id}_\delta(x))$ by Proposition 4.5. If $x =_\delta y$ then $f(\text{id}_\delta(x)) = f(\text{id}_\delta(y))$ and hence $\overline{\text{id}}_\epsilon(f(x)) \sqsubseteq f(\text{id}_\delta(y)) \sqsubseteq f(y)$. By Lemma 4.21, $f(x) =_\epsilon f(y)$, as required. $\square$

Similarly, we have:

Proposition 4.23. *For any total* $f \in (\sigma \to \gamma)$ *and any total* $x \in \sigma$*, where* $\gamma \in \{\texttt{Nat}, \texttt{Bool}\}$*,*

$$\exists \delta \in \mathbb{N}\ \forall \textit{total } y \in \sigma,\ x =_\delta y \implies f(x) = f(y).$$

5. Compact sets

The intuition behind the topological notion of compactness is that a compact set behaves, in many important respects, as if it were a set of finite cardinality — see e.g. [11]. The official topological definition, which is more obscure, says that a subset Q of a topological space is compact iff it satisfies the Heine–Borel property: any collection of open sets that covers Q has a finite subcollection that already covers Q. In order to arrive at an operational notion of compactness, we reformulate this in two stages.

(1) Any collection of open sets of a topological space can be made directed by adding the unions of finite subcollections. Hence a set Q is compact iff every directed cover of Q by open sets includes an open set that already covers Q.

(2) Considering the Scott topology on the lattice of open sets of the topological space, this amounts to saying that the collection of open sets U with $Q \subseteq U$ is Scott open in this lattice.

Thus, this last reformulation considers open sets of open sets. We take this as our definition, with "Scott open" replaced by "open" in the sense of Definition 3.5: we say that a collection $\mathcal{U}$ of open sets of a type σ is open if the collection $\{\chi_U \mid U \in \mathcal{U}\}$ is open in the function type $(\sigma \to \Sigma)$.

Lemma 5.1. *For any set Q of elements of a type σ, the following two conditions are equivalent:*

1. *The collection $\{U \text{ open} \mid Q \subseteq U\}$ is open.*
2. *There is $(\forall_Q) \in ((\sigma \to \Sigma) \to \Sigma)$ such that*
$$\forall_Q(p) = \top \iff p(x) = \top \text{ for all } x \in Q.$$

Proof. $\forall_Q = \chi_{\mathcal{U}}$ for $\mathcal{U} = \{\chi_U \mid Q \subseteq U\}$, because if $p = \chi_U$ then $Q \subseteq U \iff p(x) = \top$ for all $x \in Q$. □

Definition 5.2. We say that a set Q of elements of a type σ is *compact* if it satisfies the above equivalent conditions. In this case, for the sake of clarity, we write "$\forall x \in Q. \dots$" instead of "$\forall_Q(\lambda x. \dots)$".

Lemma 5.1(2) gives a sense in which a compact set behaves as a set of finite cardinality: it is possible to universally quantify over it in a mechanical fashion. Hence every finite set is compact. Examples of infinite compact sets will be given shortly. By Lemma 5.1(1), compact sets satisfy the "rational Heine–Borel property", because open sets are rationally Scott open:

Proposition 5.3. *If Q is compact and U_n is a rational chain of open sets with $Q \subseteq \bigcup_n U_n$, then there is $n \in \mathbb{N}$ such that already $Q \subseteq U_n$.*

Further properties of compact sets that are familiar from classical topology hold for our operational notion [8]:

Proposition 5.4.

1. *For any $f \in (\sigma \to \tau)$ and any compact set Q in σ, the set $f(Q) = \{f(x) \mid x \in Q\}$ is compact in τ.*
2. *If Q is compact in σ and R is compact in τ, then $Q \times R$ is compact in $\sigma \times \tau$.*
3. *If Q is compact in σ and V is open in τ, then*
$$N(Q,V) \stackrel{\text{def}}{=} \{f \in (\sigma \to \tau) \mid f(Q) \subseteq V\}$$
is open in $(\sigma \to \tau)$.

Proof. (1): $\forall y \in f(Q).p(y) = \forall x \in Q.p(f(x))$.
(2): $\forall z \in Q \times R.p(z) = \forall x \in Q.\forall y \in R.p(x,y)$.
(3): $\chi_{N(Q,V)}(f) = \forall x \in Q.\chi_V(f(x))$. □

The set of *all* elements of any type σ is compact, but for trivial reasons: $p(x) = \top$ holds for all $x \in \sigma$ iff it holds for $x = \bot$, by monotonicity, and hence the definition $\forall_\sigma(p) = p(\bot)$ gives a universal quantification program.

Proposition 5.5. *The total elements of* `Nat` *and* `Baire` *don't form compact sets.*

Proof. It is easy to construct $g \in (\overline{\omega} \times \texttt{Nat} \to \Sigma)$ such that $g(x,n) = \top$ iff $x > n$ for all $x \in \overline{\omega}$ and $n \in \mathbb{N}$. If the total elements $\mathbb{N}$ of `Nat` did form a compact set, then we would have $u \in (\overline{\omega} \to \Sigma)$ defined by $u(x) = \forall n \in \mathbb{N}.g(x,n)$ that would satisfy $u(k) = \bot$ for all $k \in \mathbb{N}$ and $u(\infty) = \top$ and hence would violate rational continuity. Therefore $\mathbb{N}$ is not compact in `Nat`. If the total elements of `Baire` formed a compact set, then, considering $f \in (\texttt{Baire} \to \texttt{Nat})$ defined by $f(s) = s(0)$, Proposition 5.4(1) would entail that $\mathbb{N}$ is compact in `Nat`, again producing a contradiction. □

The above proof relies on a continuity principle rather than on recursion theory. Thus, compactness of $\mathbb{N}$ in `Nat` fails even if the language includes an oracle for the Halting Problem. Taking Lemma 5.1(1) as the formulation of compactness, by Theorem 4.15 and Lemma 4.14, we have:

Proposition 5.6. *An open set is compact iff it has finite characteristic. Hence every open set is a rational union of compact open sets.*

Armed with the results that we have so far, it is easy to see that if an open set has finite characteristic then it is the upper set of a finite set of finite elements. The simplest non-trivial example of a compact set, which is a manifestation of the "one-point compactification of the discrete space of natural numbers", is given in the following proposition.

We regard function types of the form $(\texttt{Nat} \to \sigma)$ as sequence types and define "head", "tail" and "cons" constructs for sequences as follows:
$$\mathrm{hd}(s) = s(0), \qquad \mathrm{tl}(s) = \lambda i.s(i+1),$$
$$n :: s = \lambda i. \text{if } i == 0 \text{ then } n \text{ else } s(i-1).$$
We also use familiar notations such as $0^n 1^\omega$ as shorthands for evident terms such as $\lambda i. \text{if } i < n \text{ then } 0 \text{ else } 1$.

Proposition 5.7. *The set $\mathbb{N}_\infty$ of sequences of the forms $0^n 1^\omega$ and 0^ω is compact in* `Baire`.

Proof. Define, omitting the subscript $\mathbb{N}_\infty$ for $\forall$,
$$\forall(p) = p(\text{if } p(1^\omega) \wedge \forall s.p(0 :: s) \text{ then } t),$$
where t is some element of $\mathbb{N}_\infty$. More formally, $\forall = \mathrm{fix}(F)$ where
$$F(A)(p) = p(\text{if } p(1^\omega) \wedge A(\lambda s.p(0 :: s)) \text{ then } t).$$
We must show that, for any given p, $\forall(p) = \top$ iff $p(s) = \top$ for all $s \in \mathbb{N}_\infty$.

($\Leftarrow$): The hypothesis gives $p(0^\omega) = \top$. By Proposition 4.5, there is n such that already $p(\mathrm{id}_n(0^\omega)) = \top$. But $\mathrm{id}_n(0^\omega)(i) = 0$ if $i < n$ and $\mathrm{id}_n(0^\omega)(i) = \bot$ otherwise. Using this and monotonicity, a routine proof by induction on k shows that if $p(\mathrm{id}_k(0^\omega)) = \top$ then $F^k(\bot)(p) = \top$. The result hence follows from the fact that $F^k(\bot) \sqsubseteq \forall$.

($\Rightarrow$): By rational continuity, the hypothesis implies that $F^n(\bot)(p) = \top$ for some n. A routine, but slightly laborious, proof by induction on k shows that, for all q, if $F^k(\bot)(q) = \top$ then $q(s) = \top$ for all $s \in \mathbb{N}_\infty$. □

In order to construct more sophisticated examples of compact sets, we need the techniques of Section 6 below. Before that, we consider some *uniform*-continuity principles (cf. Propositions 4.22 and 4.23). Define $s \equiv_n t \iff \overline{\mathrm{id}}_n(s) = \overline{\mathrm{id}}_n(t)$ for $\overline{\mathrm{id}}_n \colon \texttt{Baire} \to \texttt{Baire}$ as in Lemma 4.21.

Lemma 5.8. *For $f \in (\sigma \to \texttt{Baire})$ total and Q a compact set of total elements of σ,*

$$\forall \epsilon \in \mathbb{N}\ \exists \delta \in \mathbb{N}\ \forall x \in Q,\ f(x) \equiv_\epsilon f(\mathrm{id}_\delta(x)).$$

Proof. For any given $\epsilon \in \mathbb{N}$, it is easy to construct a program $e \in (\texttt{Baire} \times \texttt{Baire} \to \Sigma)$ such that

(i) if $s, t \in \texttt{Baire}$ are total then $s \equiv_\epsilon t \Rightarrow e(s,t) = \top$,

(ii) for all $s, t \in \texttt{Baire}$, $e(s,t) = \top \Rightarrow s \equiv_\epsilon t$.

If we define $p(x) = e(f(x), f(x))$, then, by the hypothesis and (i), $\forall_Q(p) = \top$. By Proposition 4.5, $\forall_Q(\mathrm{id}_\delta(p)) = \top$ for some $\delta \in \mathbb{N}$, and, by Proposition 4.10, $\mathrm{id}_\delta(p)(x) = p(\mathrm{id}_\delta(x))$. It follows that $e(f(\mathrm{id}_\delta(x)), f(\mathrm{id}_\delta(x))) = \top$ for all $x \in Q$. By monotonicity, $e(f(x), f(\mathrm{id}_\delta(x))) = \top$, and, by (ii), $f(x) \equiv_\epsilon f(\mathrm{id}_\delta(x))$, as required. □

Theorem 5.9. *For $f \in (\sigma \to \texttt{Baire})$ total and Q a compact set of total elements of σ,*

$$\forall \epsilon \in \mathbb{N}\ \exists \delta \in \mathbb{N}\ \forall x, y \in Q,\ x =_\delta y \Rightarrow f(x) =_\epsilon f(y).$$

Proof. Given $\epsilon \in \mathbb{N}$, first construct $\delta \in \mathbb{N}$ as in Lemma 5.8. For $x, y \in Q$, if $x =_\delta y$ then $\overline{\mathrm{id}}_\epsilon(f(x)) = \overline{\mathrm{id}}_\epsilon(f(\mathrm{id}_\delta(x))) = \overline{\mathrm{id}}_\epsilon(f(\mathrm{id}_\delta(y))) \sqsubseteq f(y)$. By Lemma 4.21, $f(x) =_\epsilon f(y)$, as required. □

Similarly, we have:

Proposition 5.10. *For $\gamma \in \{\texttt{Nat}, \texttt{Bool}\}$, $f \in (\sigma \to \gamma)$ total and Q a compact set of total elements of σ,*

1. $\exists \delta \in \mathbb{N} \forall x \in Q,\ f(x) = f(\mathrm{id}_\delta(x))$,
2. $\exists \delta \in \mathbb{N} \forall x, y \in Q,\ x =_\delta y \implies f(x) = f(y)$.

The following is used in Section 7 below:

Definition 5.11. For f and Q as in Proposition 5.10, we refer to the least $\delta \in \mathbb{N}$ such that (1) (respectively (2)) holds as the *big* (respectively *small*) *modulus of uniform continuity* of f at Q.

6. A data language

In an operational setting, one usually adopts the same language to construct programs of a type and to express data of the same type. But consider programs that can accept externally produced streams as inputs. Because such streams are not necessarily definable in the language, it makes sense to consider program equivalence defined by quantification over more liberal "data contexts" and ask whether the same notion of program equivalence is obtained.

Definition 6.1. Let $\mathcal{P}$ be the programming language introduced in Section 2, perhaps extended with parallel features, but not with oracles, and let $\mathcal{D}$ be $\mathcal{P}$ extended with oracles. We think of $\mathcal{D}$ as a *data language* for the programming language $\mathcal{P}$. The idea is that the closed terms of $\mathcal{P}$ are *programs* and those of $\mathcal{D}$ are (higher-type) *data*. Accordingly, in this context, the notation $x \in \sigma$ means that x is a closed term of type σ in the data language. Of course, this includes the possibility that x is a program.

The following is folklore and goes back to Milner [16]:

Theorem 6.2. *For terms in $\mathcal{P}$, equivalence with respect to ground $\mathcal{P}$-contexts and equivalence with respect to ground $\mathcal{D}$-contexts coincide.*

Proof. For any oracle Ω, $\mathrm{id}_n(\Omega)$ is extensionally equivalent to some program, for both notions of equivalence. Hence for any element x of any type, $\mathrm{id}_n(x)$ is equivalent to some program. To conclude, apply Proposition 4.4. □

On the other hand, the notion of totality changes:

Theorem 6.3. *There are programs that are total with respect to $\mathcal{P}$ but not with respect to $\mathcal{D}$.*

This kind of phenomenon is again folklore. There are programs of type e.g. $\texttt{Cantor} \to \texttt{Bool}$, where $\texttt{Cantor} \stackrel{\text{def}}{=} (\texttt{Nat} \to \texttt{Bool})$, that, when seen from the point of view of the data language, map programmable total elements to total elements, but diverge at some non-programmable total inputs. The construction uses Kleene trees [5], and can be found in [8, Chapter 3.11]. This is analogous to the fact that totality with respect to $\mathcal{P}$ also disagrees with totality with respect to denotational models. A proof for the Scott model can be found in [22]. For the intriguing relationship between totality in the Scott model with sequential computation, see [17].

7. Sample applications

We use the data language $\mathcal{D}$ to formulate specifications of programs in the programming language $\mathcal{P}$. As in Section 6, the notation $x \in \sigma$ means that x is a closed term of type σ in $\mathcal{D}$. This is compatible with the notation of Sections 3–5 by taking $\mathcal{D}$ as the underlying language for them. Again maintaining compatibility, we take the notions of totality, open set and compact set with respect to $\mathcal{D}$. To indicate that openness or compactness of a set is witnessed by a program rather than just an element of the data language, we say *programmably* open or compact.

As for the Baire type, we think of the elements of the Cantor type as sequences, and, following topological tradition, in this context we identify the booleans true and false with the numbers 0 and 1 (it doesn't matter in which order). The following is our main tool in this section:

Theorem 7.1. *The total elements of the Cantor type form a programmably compact set.*

Proof. This is proved and discussed in detail in [8, Chapter 3.11], and hence we only provide the construction of the universal quantification program, with one minor improvement. We recursively define $\forall\colon (\texttt{Cantor} \to \Sigma) \to \Sigma$ by

$$\forall(p) = p(0 :: \text{if } \forall s.p(0 :: s) \wedge \forall s.p(1 :: s) \text{ then } t),$$

where t is some programmable total element of `Cantor`. The correctness proof for this program is similar to that of Proposition 5.7, but involves an invocation of König's Lemma. $\square$

Remark 7.2. If the data language is taken to be $\mathcal{P}$ itself, Theorem 7.1 fails for the same reason that leads to Theorem 6.3 [8, Chapter 3.11]. Of course, the above program $\forall\colon (\texttt{Cantor} \to \Sigma) \to \Sigma$ can still be written down. But it no longer satisfies the required specification given in Lemma 5.1(2). In summary, it is easier to universally quantify over *all* total elements of the Cantor type than just over the *programmable* ones, to the extent that the former can be achieved by a program but the latter cannot.

Interestingly, the programmability conclusion of Theorem 7.1 is not invoked for the purposes of this section, because we only apply compactness to get uniform continuity.

The following theorem is due to Berger [6], with domain-theoretic denotational specification and proof. As discussed in the introduction, the purpose of this section is to illustrate that such specifications and proofs can be directly understood in our operational setting, and, moreover, apply to *sequential* programming languages.

Theorem 7.3. *There is a total program*

$$\varepsilon\colon (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Cantor}$$

s.t. for any total $p \in (\texttt{Cantor} \to \texttt{Bool})$*, if* $p(s) = \mathrm{true}$ *for some total* $s \in \texttt{Cantor}$*, then* $\varepsilon(p)$ *is such an* s*.*

Proof. Define

$$\varepsilon(p) = \text{if}\, p(0 :: \varepsilon(\lambda s.p(0 :: s)))\ \ \text{then}\ \ 0 :: \varepsilon(\lambda s.p(0 :: s)) \\ \text{else}\ \ 1 :: \varepsilon(\lambda s.p(1 :: s)).$$

The required property is established by induction on the big modulus of uniform continuity of a total element $p \in (\texttt{Cantor} \to \texttt{Bool})$ at the set of total elements, using the fact that if p has modulus $\delta+1$ then $\lambda s.p(0 :: s)$ and $\lambda s.p(1 :: s)$ have modulus δ, and that when p has modulus zero, $p(\bot)$ is total and hence p is constant. $\square$

This gives rise to universal quantification for boolean-valued rather than Sierpinski-valued predicates:

Corollary 7.4. *There is a total program*

$$\forall\colon (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Bool}$$

such that for every total $p \in (\texttt{Cantor} \to \texttt{Bool})$*,*

$$\forall(p) = \mathrm{true} \Leftrightarrow p(s) = \mathrm{true}\ \textit{for all total}\ s \in \texttt{Cantor}.$$

Proof. First define $\exists\colon (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Bool}$ by $\exists(p) = p(\varepsilon(p))$ and then define $\forall(p) = \neg\exists s.\neg p(s)$. $\square$

Corollary 7.5. *The function type* $(\texttt{Cantor} \to \texttt{Nat})$ *has decidable equality for total elements.*

Proof. Define a program

$$(==)\colon (\texttt{Cantor} \to \texttt{Nat}) \times (\texttt{Cantor} \to \texttt{Nat}) \to \texttt{Bool}$$

by $(f == g) = \forall$ total $s \in \texttt{Cantor}. f(s) == g(s)$. $\square$

Simpson [24] applied Corollary 7.4 to develop surprising sequential programs for computing integration and supremum functionals $([0, 1] \to \mathbb{R}) \to \mathbb{R}$, with real numbers represented as infinite sequences of digits. The theory developed here copes with that, again allowing a direct operational translation of the original denotational development. For lack of space to introduce the necessary background on real number-computation, we illustrate the main idea by reformulating the development of the supremum functional, with the closed unit interval and the real line replaced by the Cantor and Baire types, and with the natural order of the reals replaced by the lexicographic order on sequences.

The *lexicographic order* on the total elements of the Baire type is defined by $s \leq t$ iff whenever $s \neq t$, there is $n \in \mathbb{N}$ with $s(n) < t(n)$ and $s(i) = t(i)$ for all $i < n$.

Lemma 7.6. *There is a total program*

$$\max\colon \texttt{Baire} \times \texttt{Baire} \to \texttt{Baire}$$

such that

1. $\max(s,t)$ *is the maximum of* s *and* t *in the lexicographic order for all total* $s, t \in \texttt{Baire}$*, and*
2. $(s, t) =_\epsilon (s', t') \Rightarrow \max(s, t) =_\epsilon \max(s', t')$ *for all* $s, t, s', t' \in \texttt{Baire}$ *(total or not) and all* $\epsilon \in \mathbb{N}$*.*

Proof. Define

$$\begin{aligned} \max(s,t) \quad = \quad & \text{if}\, \mathrm{hd}(s) == \mathrm{hd}(t) \\ & \text{then}\ \mathrm{hd}(s) :: \max(\mathrm{tl}(s), \mathrm{tl}(t)) \\ & \text{else if}\ \mathrm{hd}(s) > \mathrm{hd}(t)\ \text{then}\ s\ \text{else}\ t. \end{aligned}$$

The easy details of the correctness proof are omitted. $\square$

Theorem 7.7. *There is a total program*

$$\sup\colon (\texttt{Cantor} \to \texttt{Baire}) \to \texttt{Baire}$$

such that for every total $f \in (\texttt{Cantor} \to \texttt{Baire})$*,*

$$\sup(f) = \sup\{f(s) \mid s \in \texttt{Cantor}\ \textit{is total}\},$$

where the supremum is taken in the lexicographic order.

Proof. Let $t \in \texttt{Cantor}$ be a programmable total element and define

$$\begin{aligned} \sup(f) = {} & \text{let}\ h = \mathrm{hd}(f(t))\ \text{in} \\ & \quad \text{if}\ \forall\ \text{total}\ s \in \texttt{Cantor}.\, \mathrm{hd}(f(s)) == h \\ & \quad \text{then}\ h :: \sup(\lambda s.\, \mathrm{tl}(f(s))) \\ & \quad \text{else}\ \max(\sup(\lambda s.f(0 :: s)), \sup(\lambda s.f(1 :: s))), \end{aligned}$$

where "let $x = \ldots$ in M" stands for "$(\lambda x.M)(\ldots)$".

One shows by induction on $n \in \mathbb{N}$ that, for every total $f \in (\texttt{Cantor} \to \texttt{Baire})$,

$$\sup(f) =_n \sup\{f(s) \mid s \in \texttt{Cantor}\ \text{is total}\}.$$

The base case is trivial. For the induction step, one proceeds by a further induction on the small modulus of uniform continuity of $\mathrm{hd} \circ f\colon \texttt{Cantor} \to \texttt{Nat}$ at the total elements of `Cantor`, crucially appealing to the non-expansiveness condition given by Lemma 7.6(2). One uses the facts that if $\mathrm{hd} \circ f$ has modulus $\delta + 1$ then $\mathrm{hd} \circ \lambda s.f(0 :: s)$ and $\mathrm{hd} \circ \lambda s.f(1 :: s)$ have modulus δ, and that if $\mathrm{hd} \circ f$ has modulus 0 then $\mathrm{hd}(f(s)) = \mathrm{hd}(f(t))$ for all total s and t. $\square$

Theorems 7.3 and 7.7 rely on the compactness of the total elements of the Cantor type. Arguments similar to that of Proposition 5.5 show that these two theorems fail if the Cantor type is replaced by the Baire space.

8. Open problems and further developments

The Tychonoff theorem in classical topology states that a product of arbitrarily many compact spaces is compact. A proof that this holds in a computational setting for countably many compact spaces is developed in [8, Chapter 13]. Moreover, the given implementation is sequential. However, the proposed proof is for the specification of the program interpreted in the Scott model. At the time of writing, we are not able to apply our techniques to derive a correctness proof of the program for an interpretation of the specification in the sequential data language considered here.

Our use of sequence types $(\mathtt{Nat} \to \sigma)$ can be easily replaced by lazy lists by applying the bisimulation techniques of [9] to prove the correctness of evident programs that implement the SFP property for lazy lists. There is no difficulty in developing the results of this paper in a call-by-value setting, and we believe we can also handle recursive types. But computational features such as state and control, and non-determinism and probability seem to pose genuine challenges.

In the presence of probability or of abstract data types for real numbers, types won't be algebraic in general and hence a binary notion of finiteness, analogous to the way-below relation in classical domain theory, needs to be developed.

The avoidance of syntactic manipulations suggests that the theory worked out in this paper could be developed in a general axiomatic framework rather than just term models. In particular, this would make our results available to models that are not constructed from domain-theoretic or topological data, e.g. games models.

Acknowledgements. The impossibility of a constructive proof of Theorem 4.15 (Remark 4.17(2)) was found together with Vincent Danos during a visit to our institution. We also benefited from valuable feedback by Achim Jung, Paul B. Levy, Andy Pitts, Uday Reddy, Alex Simpson, Thomas Streicher and Steve Vickers.

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inform. and Comput.*, 163(2):409–470, 2000.

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

[3] R.M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. CUP, 1998.

[4] S. Awodey, L. Birkedal, and D.S. Scott. Local realizability toposes and a modal logic for computability. *Math. Structures Comput. Sci.*, 12(3):319–334, 2002.

[5] M.J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

[6] U. Berger. *Totale Objekte und Mengen in der Bereichstheorie*. PhD thesis, Mathematisches Institut der Universität München, 1990.

[7] U. Berger. Computability and totality in domains. *Math. Structures Comput. Sci.*, 12(3):281–294, 2002.

[8] M.H. Escardó. Synthetic topology of data types and classical spaces. *Electron. Notes Theor. Comput. Sci.*, 87:21–156, 2004.

[9] A.D. Gordon. Bisimilarity as a theory of functional programming. *Theoret. Comput. Sci.*, 228(1-2):5–47, 1999.

[10] C.A. Gunter. *Semantics of Programming Languages—Structures and Techniques*. The MIT Press, 1992.

[11] E. Hewitt. The rôle of compactness in analysis. *Amer. Math. Monthly*, 67:499–516, 1960.

[12] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Inform. and Comput.*, 163(2):285–408, 2000.

[13] A. Jung. Talk at the Workshop on Full abstraction of PCF and related Languages, BRICS institute, Aarhus, 1995.

[14] R. Loader. Finitary PCF is not decidable. *Theoret. Comput. Sci.*, 266(1-2):341–364, 2001.

[15] I.A. Mason, S.F. Smith, and C.L. Talcott. From operational semantics to domain theory. *Inform. and Comput.*, 128(1):26–47, 1996.

[16] R. Milner. Fully abstract models of typed λ-calculi. *Theoret. Comput. Sci.*, 4(1):1–22, 1977.

[17] D. Normann. Computability over the partial continuous functionals. *J. Symbolic Logic*, 65(3):1133–1142, 2000.

[18] A.M. Pitts. A note on logical relations between semantics and syntax. *Logic Journal of the Interest Group in Pure and Applied Logics*, 5(4):589–601, July 1997.

[19] A.M. Pitts. Operationally-based theories of program equivalence. In *Semantics and logics of computation (Cambridge, 1995)*, volume 14 of *Publ. Newton Inst.*, pages 241–298. CUP, 1997.

[20] A.M. Pitts. Operational semantics and program equivalence. In G. Barthe, P. Dybjer, and J. Saraiva, editors, *Applied Semantics, Advanced Lectures*, volume 2395 of *Lec. Not. Comput. Sci., Tutorial*, pages 378–412. Springer, 2002.

[21] G.D. Plotkin. LCF considered as a programming language. *Theoret. Comput. Sci.*, 5(1):223–255, 1977.

[22] G.D. Plotkin. Full abstraction, totality and PCF. *Math. Structures Comput. Sci.*, 9(1):1–20, 1999.

[23] D.S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoret. Comput. Sci.*, 121:411–440, 1993. Reprint of a 1969 manuscript.

[24] A. Simpson. Lazy functional algorithms for exact real functionals. *Lec. Not. Comput. Sci.*, 1450:323–342, 1998.

[25] M.B. Smyth. Power domains and predicate transformers: a topological view. volume 154 of *Lec. Not. Comput. Sci.*, pages 662–675, 1983.

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

[27] K. Weihrauch. *Computable analysis*. Springer, 2000.