Operational domain theory and topology of sequential programming languages

Martín Escardó   Weng Kin Ho
School of Computer Science, University of Birmingham, UK

November 18, 2006

Abstract

A number of authors have exported domain-theoretic techniques from denotational semantics to the operational study of contextual equivalence and order. 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, in addition to languages with parallel features.

1 Introduction

Domain theory and topology in programming language semantics have been applied to manufacture and study *denotational* models, starting with the Scott model of PCF [32]. 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 [29]. 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 side-stepping denotational semantics and reformulating domain-theoretic and topological notions directly in terms of programming concepts, interpreted in an operational way.

Regarding domain theory [3, 11], 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.

We work with a restricted kind of increasing chain because we must: Dag Normann [25] has shown that, even in the presence of oracles (see below), increasing chains in the contextual order fail to have suprema in general. A counter-example is given for type level 3. On the other hand, it is known that rational chains always have suprema, even in the absence of oracles — see e.g. [27].

Regarding topology [23, 35], 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 [2, 17], 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 [29, 10]. But we don't rely on these facts.

We illustrate the scope and flexibility of the theory by applying our conclusions to prove the correctness of various non-trivial programs that manipulate infinite data. We take one such example from [33]. 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.

1.1 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, Talcott [21] and Sands (see Pitts [27] for references). Already in [21], 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 4.16).

The idea that topological techniques can be directly understood in terms of operational semantics, and, moreover, are applicable to sequential languages, is due to the first-named author [10]. In particular, we have taken our operational notion of compactness and some material about it from that reference. A 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 [10] and is related to relative realizability [5] and TTE [37].

1.2 Organization

Section 2: Language, oracles, extensionality, monotonicity and rational chains.
Section 3: Rational chains, open sets and continuity principles.

2 Preliminaries

2.1 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 [13, 27]. Other possibilities are briefly discussed in Section 8.2.

For clarity of exposition, we explicitly 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 [31]). The type Σ will have elements $\perp$ (non-terminating computation) and $\top$ (terminating computation). Intuitively, we think of programs of type $\overline{\omega}$ as clocks that either tick for ever or else tick finitely often and then fail (see Section 2.5 below for a precise mathematical statement). What is relevant for our purposes is that, for any type σ, functions $\sigma \to \Sigma$ will correspond to semi-decidable or open sets of elements of σ, and functions $\overline{\omega} \to \sigma$ will correspond to certain ascending chains of elements of σ in the contextual order (in fact, precisely the rational chains, to be defined below). In this sense, Σ will classify open sets (this belongs to the realm of topology) and $\overline{\omega}$ will co-classify rational chains (this belongs to the realm of domain theory).

Formally, we have the following term-formation rules for these two types:

(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.

Notice that there is no "else" clause in the above construction. 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 $\bot_\sigma = \mathrm{fix}\, x.x$, 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 $\mathrm{fix}\, x.f(x)$.

2.2 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.

2.3 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*.

2.4 Notation for contextual equivalence and (pre)order

We write $M = N$ and $M \sqsubseteq N$ to denote contextual equivalence and order of terms of the same type.

2.5 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 $\bot$ and $\top$ of Σ are contextually ordered by $\bot \sqsubseteq \top$, they 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 $\bot$ as "unobservable false".

The elements of $\overline{\omega}$. We denote by ∞ the element $\operatorname{fix} x.x + 1$ of $\overline{\omega}$, and, by an abuse of notation, for $n \in \mathbb{N}$ we write n to denote the element $\operatorname{succ}^n(\bot)$ of $\overline{\omega}$, where $\operatorname{succ}(x) = x + 1$. The elements $0, 1, 2, \ldots, n, \ldots, \infty$ of $\overline{\omega}$ are all contextually inequivalent, and any element of $\overline{\omega}$ is contextually equivalent to one of them. They are contextually ordered by

$$0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \ldots \sqsubseteq n \sqsubseteq \ldots \sqsubseteq \infty.$$

C.f. Section 3.1 below. Notice that $0 - 1 = 0$, $(x + 1) - 1 = x$, $(0 > 0) = \bot$ and $(x + 1 > 0) = \top$ hold for $x \in \overline{\omega}$. In particular, $\infty - 1 = \infty$ and $(\infty > 0) = \top$.

2.6 Extensionality and monotonicity

Contextual equivalence is a congruence: for any $f, g \in (\sigma \to \tau)$ and $x, y \in \sigma$,

if $f = g$ and $x = y$ then $f(x) = g(y)$.

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 $f(x) \sqsubseteq g(y)$.

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, $\bot_\sigma$ is the least element of σ.

2.7 Rational chains

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

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

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

$$x_n = h(g^n(\bot)).$$

2.8 Proofs

The facts stated in this background section are all well known. The extensionality, monotonicity and rational-chain principles follow directly from Milner's construction [22]. Even though full abstraction of the Scott model fails for sequential languages, proofs exploiting computational adequacy are possible [18] (see [26]). Proofs using game semantics can be found in [2, 17], and operational proofs can be found in [27, 28] (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 [21]. 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 [12] or [27] (see [16]).

3 Rational chains and open sets

We begin by developing fundamental order-theoretic and topological properties of the types of our language.

3.1 Order

By the results recalled in the previous section, every rational chain is increasing and has a least upper bound in the contextual order.

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

$$l(\infty) = \bigsqcup_n l(n).$$

Proof. $n = \mathrm{succ}^n(\bot)$ and $\infty = \mathrm{fix}\,\mathrm{succ}$. □

Moreover, this is the "generic rational chain" with "generic least upper bound ∞" 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 $n \in \mathbb{N}$,*

$$x_n = l(n),$$

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 $x_n = h(g^n(\bot))$, recursively define

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

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

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

This easy observation seems to be new.

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 $F(f) = f(x)$. □

3.2 Topology

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. Notice that in this case U is closed under contextual equivalence, i.e., any element equivalent to a member of U is also a member of U. For example, the subset $\{\top\}$ of Σ is open, as its characteristic function is the identity, but $\{\bot\}$ is not, because a characteristic function would have to send $\bot$ to $\top$ and $\top$ to $\bot$, violating monotonicity.

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*

1. *finite intersections and*
2. *rational unions.*

Proof. (1): $\chi_{\bigcap \emptyset}(x) = \top$ and $\chi_{U \cap V}(x) = \chi_U(x) \wedge \chi_V(x)$, where $\wedge$ is defined as

$$p \wedge q = \text{if } p \text{ then } q.$$

(2): 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 a topology in the classical sense. Recall that a collection of subsets of a given set X is called a topology on X if it is closed under the formation of finite intersections and arbitrary unions. The thrust of this work is that a wealth of classical theorems on topological spaces, in particular Scott domains under the Scott topology, happen to hold for program types, even though the types of sequential languages don't form topological spaces 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. ($\Uparrow$): $\chi_{\bigcup \emptyset}(x) = \bot$ and $\chi_{U \cup V}(x) = \chi_U(x) \vee \chi_V(x)$.

($\Downarrow$): The sets $U = \{(p,q) \mid p = \top\}$ 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 [10, 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. [27, 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. (1): By monotonicity of χ_U.

(2) By rational continuity of χ_U : If $\bigsqcup x_n \in U$ then $\top = \chi_U(\bigsqcup_n x_n) = \bigsqcup \chi_U(x_n)$ and hence $\top = \chi_U(x_n)$ for some n, i.e., $x_n \in U$. □

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 if and only if any attempt to build b as the least upper bound 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 $b \sqsubseteq x_n$.

4.1 Algebraicity

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 subsection. 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. $\square$

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. $\square$

Of course, the above holds with contextual equivalence replaced by contextual order. Another consequence of Theorem 4.2 is a third continuity principle, which is reminiscent of the ϵ–δ formulation of continuity in real analysis (cf. Section 4.4), and says that finite parts of the output of a program depend only on finite parts of the input:

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 $b \sqsubseteq x$ such that already $c \sqsubseteq f(b)$.*

Proof. By Theorem 4.2, x is the least upper bound of a rational chain b_n of finite elements. By rational continuity, $c \sqsubseteq \bigsqcup_n f(b_n)$, and, by finiteness of c, there is n with $c \sqsubseteq f(b_n)$. $\square$

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 $\top$. $\square$

In order to prove Theorem 4.2, we invoke the following concepts (see e.g. [3]):

Definition 4.7.

1. A *deflation* on a type σ is an element of type $(\sigma \to \sigma)$ that
   (a) is below the identity of σ, and
   (b) 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 at least one 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 particular, because id_n is idempotent, $\mathrm{id}_n(x)$ is finite and hence any $x \in \sigma$ is the least upper bound 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, modulo contextual equivalence, the set $\{\mathrm{id}_n(x_i) \mid i \in \mathbb{N}\}$ is finite and hence has a maximal element, which is its least upper bound. 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 $b \sqsubseteq x_i$, by transitivity, 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* $\mathrm{id}_n^\sigma \in (\sigma \to \sigma)$ *can be chosen for each type* σ *in such a way that*

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

Item 1 gives an expected conclusion:

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 $k \geq n$. (The inequality $x = \mathrm{id}_n(x) \sqsubseteq \mathrm{id}_k(x)$ holds because id_i is an increasing chain, and the inequality $\mathrm{id}_k(x) \sqsubseteq x$ holds because id_k is a deflation.)

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. □

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

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

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 chain* $\mathrm{id}_n^\sigma \stackrel{\mathrm{def}}{=} \mathrm{d}^\sigma(n)$ *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. By induction on n, we have that $\mathrm{d}^{\overline{\omega}}(n)(y) = \min(n, y)$ for every $n \in \mathbb{N}$. Hence $\mathrm{d}^{\overline{\omega}}(n)$ is idempotent and below the identity, and has image $\{0, 1, \ldots, n\}$. Now $\mathrm{d}^{\overline{\omega}}(\infty)(k) = \bigsqcup_n \mathrm{d}^{\overline{\omega}}(n)(k) = \bigsqcup_n \min(n, k) = k$ for $k \in \mathbb{N}$. 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.

4.2 Topological characterization of finiteness

We first need some preliminary material.

Definition 4.14. We say that an open set in σ has *finite characteristic* if its characteristic function is a finite element of the function type $(\sigma \to \Sigma)$.

It is clear that every open set is the union of a rational chain of open sets of finite characteristic. The following spells this out in more detail.

Lemma 4.15. *For any open set* U *in* σ *and any* $n \in \mathbb{N}$, *let*

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

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 $\mathrm{id}_n^{\sigma\to\Sigma}$ has finite image modulo contextual equivalence.

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

Theorem 4.16. *An element* $b \in \sigma$ *is finite if and only if the set*

$$\uparrow b \stackrel{\mathrm{def}}{=} \{x \in \sigma \mid b \sqsubseteq x\}$$

is open.

From the point of view of classical domain theory, this is a tautology: b is finite, by definition, if every directed set with supremum above b already has an element above b, which, again by definition, means that the set $\uparrow b$ is Scott open. But the situation here is entirely different. Although one direction of the proof of the above theorem amounts to this observation, the other has to be non-trivial: we know that openness implies rational Scott openness, but there is no reason to suspect that the converse holds in general — this is corroborated by Proposition 4.20 below.

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 4.15(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 4.15(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. $b \sqsubseteq x_n$. □

Hence the open sets $\uparrow b$ with b finite form a base of the (rational) topology:

Corollary 4.17. *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). □

Remark 4.18.

1. Notice that the proof of Theorem 4.16($\Rightarrow$) is not constructive, in a strong sense. 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. So, although the theorem shows that the required program $\chi_{\uparrow b}$ exists, it doesn't explicitly exhibits it.

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 [19].

Proposition 4.19. *If an open set U has finite characteristic then*

$$U = \uparrow F \stackrel{\text{def}}{=} \bigcup\{\uparrow b \mid b \in F\}$$

for some set F of finite cardinality consisting of finite elements.

Proof. By Lemma 4.15, if U has finite characteristic then there is n with $U = \mathrm{id}_n^{-1}(U)$. By construction of id_n, the set $F = \mathrm{id}_n(U)$ has finite cardinality and consists of finite elements. Now, if $x \in U$, then $x \in \uparrow F$ because x is above $\mathrm{id}_n(x)$. Conversely, if $x \in \uparrow F$, then $\mathrm{id}_n(u) \sqsubseteq x$ for some $u \in U$; but $\mathrm{id}_n(u) \in U$ because $U = \mathrm{id}_n^{-1}(U)$, and hence $x \in U$ because open sets are upper. □

The converse fails in a sequential setting:

Proposition 4.20. *The following are equivalent.*

1. *For every set F of finite cardinality consisting of finite elements of the same type, the set $\uparrow F$ is open.*

2. *There is $(\vee)\colon \Sigma \times \Sigma \to \Sigma$ such that*

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

Proof. ($\Uparrow$): Use Proposition 3.7($\Uparrow$).

($\Downarrow$): In the proof of Proposition 3.7($\Downarrow$), notice that $U = \uparrow(\top, \bot)$ and $V = \uparrow(\bot, \top)$ and observe that for $F = \{(\top, \bot), (\bot, \top)\}$ we have $\uparrow F = U \cup V$. □

4.3 Density of the total elements

We now develop an operational version of the Kleene–Kreisel density theorem [9].

Definition 4.21. (Hereditary) totality is defined by induction on types as follows:

1. An element of ground type is *total* iff it is maximal in the contextual order.
2. An element $f \in (\sigma \to \tau)$ is *total* iff $f(x) \in \tau$ is total whenever $x \in \sigma$ is total.
3. An element of type $(\sigma \times \tau)$ is total iff its projections onto σ and τ are total, or, equivalently, it is contextually equivalent to an element (x, y) with $x \in \sigma$ and $y \in \tau$ 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 4.22. *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 $y_1, \dots, y_n$. □

Theorem 4.23. *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

$$F^\tau \colon \overline{\omega} \to ((\tau \to \tau) \to \tau), \qquad G^\tau_n \colon (\tau \to \tau) \to \tau$$

by

$$F(x)(f) = \text{if } x > 0 \text{ then } f(F(x-1)(f)), \qquad G_n(f) = f^n(t)$$

for some chosen $t \in \tau$ 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 4.22, and $(\lambda x.\tilde{b})(n) \sqsubseteq \hat{b}$ and hence $b \sqsubseteq \hat{b}$ by transitivity. □

4.4 ϵ–δ formulation of continuity

We now formulate continuity in the $\epsilon - \delta$ style of real analysis. The following, used in the proof of Lemma 6.2 below, says that in order to know $f(x)$ with a given finite precision ϵ, it is enough to know x with some sufficiently sharp finite precision δ.

Lemma 4.24. *For any* $f \in (\sigma \to \tau)$*, any* $x \in \sigma$ *and any* $\epsilon \in \mathbb{N}$*, there exists* $\delta \in \mathbb{N}$ *such that* $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(\mathrm{id}_\delta(x)))$.

Proof. Since $\mathrm{id}_\epsilon(f(x)) = \bigsqcup_\delta \mathrm{id}_\epsilon \circ f \circ \mathrm{id}_\delta(x)$, it follows from the finiteness of $\mathrm{id}_\epsilon(f(x))$ that there exists $\delta \in \mathbb{N}$ such that $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(\mathrm{id}_\delta(x)))$. $\square$

Although this is reminiscent of the ϵ–δ notion of continuity in analysis, and rather useful in practice, it is not quite the same, as the definition in analysis involves the notion of closeness of two points, articulated by a notion of distance. Given a distance function d and points x and y, one says that x and y are ϵ-close, where ϵ is a positive real number, if $d(x, y) < \epsilon$. Then continuity of a function f at a point x means that for every precision $\epsilon > 0$ with which we wish to know $f(x)$, there is a sufficiently sharp precision $\delta > 0$ such that for every y that is δ-close to x, we have that $f(y)$ is ϵ-close to $f(x)$. Hence $f(y)$ is a sufficiently precise approximation of $f(x)$, so that it is not necessary to know x exactly in order to get an ϵ-precise approximation of $f(x)$.

Our next goal is to develop an analogue of this situation. We replace the closeness relation $d(x, y) < \epsilon$, where x and y are points and $\epsilon > 0$ is a real number, by the relation $x =_\epsilon y$, where x and y are elements of a type of our language and ϵ is a natural number:

$$x =_\epsilon y \iff \mathrm{id}_\epsilon(x) = \mathrm{id}_\epsilon(y).$$

But notice an important difference: in analysis, the smaller the real number $\epsilon > 0$ is, the closer x and y are when $d(x, y) < \epsilon$. Here, on the other hand, the bigger the natural number ϵ is, the closer the two elements are when $x =_\epsilon y$. If one thinks of $\mathrm{id}_\epsilon(x)$ as the truncation of the possibly infinite object x to finite precision ϵ, then $x =_\epsilon y$ means that a precision higher than ϵ is needed to distinguish x and y.

Just as in classical topology and functional analysis, in our setting it is not the case that our continuous functions are continuous in the ϵ-δ sense for all types. So we consider special types of interest. We refer to the function type $(\mathtt{Nat} \to \mathtt{Nat})$ as the *Baire type* and denote it by `Baire`:

$$\mathtt{Baire} = (\mathtt{Nat} \to \mathtt{Nat}).$$

We think of this as the type of possibly partial sequences of natural numbers. Then the set of *total* elements of `Baire` is an operational manifestation of the *Baire space* of classical topology. The following technical lemma is easily proved:

Lemma 4.25. *Define* $\overline{\mathrm{id}}_\epsilon : \texttt{Baire} \to \texttt{Baire}$ *by*

$$\overline{\mathrm{id}}_\epsilon(s) = \lambda i.\, \text{if } i < \epsilon \text{ then } s(i) \text{ else } \bot.$$

Then $\overline{\mathrm{id}}_\epsilon(s)$ *is finite and above* $\mathrm{id}_\epsilon(s)$*, and if* $s, t \in \texttt{Baire}$ *are total then for all* $\epsilon \in \mathbb{N}$,

$$\overline{\mathrm{id}}_\epsilon(s) \sqsubseteq t \implies s =_\epsilon t.$$

Theorem 4.26. *For any total* $f \in (\sigma \to \texttt{Baire})$ *and any total* $x \in \sigma$,

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

Proof. Because $\overline{\mathrm{id}}_\epsilon(f(x))$ is finite and below $f(x)$, there is δ such that already $\overline{\mathrm{id}}_\epsilon(f(x)) \sqsubseteq f(\mathrm{id}_\delta(x))$ by Proposition 4.5. If $x =_\delta y$ then $f(\mathrm{id}_\delta(x)) = f(\mathrm{id}_\delta(y))$ and hence $\overline{\mathrm{id}}_\epsilon(f(x)) \sqsubseteq f(\mathrm{id}_\delta(y)) \sqsubseteq f(y)$. By Lemma 4.25, $f(x) =_\epsilon f(y)$, as required. □

Similarly, we have:

Theorem 4.27. *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 classical 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. [14]. The official 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.

5.1 Operational formulation of the notion of compactness

In order to arrive at an operational notion of compactness, we reformulate the above definition 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:

Definition 5.1. 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 5.2. *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 5.3. 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 "$\forall x \in Q. \ldots$" instead of "$\forall_Q(\lambda x. \ldots)$".

Lemma 5.2(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.

5.2 Basic classical properties

By Lemma 5.2(1), compact sets satisfy the "rational Heine–Borel property", because open sets are rationally Scott open:

Proposition 5.4. *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 [10]:

Proposition 5.5.

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*
$$N(Q, V) \overset{\text{def}}{=} \{f \in (\sigma \to \tau) \mid f(Q) \subseteq V\}$$
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): $\chi_{N(Q,V)}(f) = \forall x \in Q.\chi_V(f(x))$. □

5.3 First examples and counter-examples

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 5.6. *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 5.5(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. The second part of the following says that the types of our language are "rationally spectral" spaces:

Theorem 5.7. *An open set is compact iff it has finite characteristic. Hence every open set is a rational union of compact open sets.*

Proof. By Proposition 5.2(1), an open set V is compact if and only if the collection $\{U \text{ open } \mid V \subseteq U\}$ is open, if and only if $\{\chi_U \mid U \text{ open and } V \subseteq U\}$ is open, if and only if the set $\uparrow \chi_V$ is open. It then follows from Theorem 4.16 that this is equivalent to χ_V being finite, i.e. V having finite characteristic. The last part of the proposition then follows from Lemma 4.15 □

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:

$$\begin{array}{rcl} \mathrm{hd}(s) & = & s(0), \\ \mathrm{tl}(s) & = & \lambda i.s(i+1), \\ n :: s & = & \lambda i.\,\mathrm{if}\, i == 0 \text{ then } n \text{ else } s(i-1). \end{array}$$

We also use familiar notations such as

$$0^n 1^\omega$$

as shorthands for evident terms such as

$$\lambda i.\,\mathrm{if}\, i < n \text{ then } 0 \text{ else } 1.$$

Theorem 5.8. *The set $\mathbb{N}_\infty$ of sequences of the forms 0^n1^ω and 0^ω is compact in the type* `Baire`.

Proof. Define, omitting the subscript $\mathbb{N}_\infty$ for $\forall$,

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

where t is some element of $\mathbb{N}_\infty$. More formally, $\forall = \mathrm{fix}(F)$ where

$$F(A)(p) = p(\mathrm{if}\, p(1^\omega) \wedge A(\lambda s.p(0 :: s)) \text{ then } t).$$

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.

5.4 Uniform continuity

We now show that certain programs are *uniformly* continuous on certain sets (cf. Theorems 4.26 and 4.27). Recall from Section 4.4 that we defined, for elements x and y of the same type, and any natural number ϵ,

$$x =_\epsilon y \iff \mathrm{id}_\epsilon(x) = \mathrm{id}_\epsilon(y).$$

For technical purposes, we now also define

$$x \equiv_\epsilon y \iff \overline{\mathrm{id}}_\epsilon(x) = \overline{\mathrm{id}}_\epsilon(y).$$

where $\overline{\mathrm{id}}_\epsilon : \mathtt{Baire} \to \mathtt{Baire}$ is defined as in Lemma 4.25:

$$\overline{\mathrm{id}}_\epsilon(s) = \lambda i.\, \mathrm{if}\ i < \epsilon\ \mathrm{then}\ s(i)\ \mathrm{else}\ \bot.$$

Lemma 5.9. *For $f \in (\sigma \to \mathtt{Baire})$ 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

$$e \in (\mathtt{Baire} \times \mathtt{Baire} \to \Sigma)$$

such that

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

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

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 5.10. *For $f \in (\sigma \to \mathtt{Baire})$ 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 5.9. 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 4.25, $f(x) =_\epsilon f(y)$, as required. □

Similarly, we have:

Theorem 5.11. *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 5.12. For f and Q as in Theorem 5.11, 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. (In the literature, e.g.[33], these are sometimes referred to as the *intensional* and *extensional* moduli of continuity respectively.)

5.5 Compact saturated sets

The remainder of the paper doesn't depend on the material of this subsection. In classical domain theory and topology, among all compact sets, the saturated ones play a distinguished role. Here we analyse the extent to which classical results about compact saturated sets generalize to our operational setting. The main result is that, as is the case for algebraic (and more generally, continuous) domains in classical domain theory, every compact saturated set of elements of any type is an intersection of upper sets of finite sets of finite elements.

Definition 5.13. The *saturation* of a subset S of a type σ is defined to be the intersection of its open neighbourhoods and is denoted by $\mathrm{sat}(S)$, i.e.,

$$\mathrm{sat}(S) = \bigcap \{U \text{ open } | S \subseteq U\}.$$

A set S is said to be *saturated* if $S = \mathrm{sat}(S)$.

In classical domain theory, a set is saturated in this sense if and only if it is an upper set. As we shall see shortly, in our sequential operational setting, every saturated set is an upper set, but the converse fails in general.

Proposition 5.14. *Let S be a subset of a type.*

1. *$S \subseteq U$ for U open if and only if $\mathrm{sat}(S) \subseteq U$.*
2. *$\uparrow S \subseteq \mathrm{sat}(S)$.*
3. *$\mathrm{sat}(S)$ is saturated.*
4. *$\mathrm{sat}(S)$ is the largest set with the same neighbourhoods as S.*

Proof. Clearly $S \subseteq \mathrm{sat}(S)$. Hence $\mathrm{sat}(S) \subseteq U$ implies $S \subseteq U$. Conversely, if $S \subseteq U$, then by definition, $\mathrm{sat}(S) \subseteq U$. So (1) holds. If $t \in \uparrow S$, then $s \sqsubseteq t$ for some $s \in S$. Hence t belongs to every neighbourhood of S, and so to $\mathrm{sat}(S)$. Therefore $\uparrow S \subseteq \mathrm{sat}(S)$, i.e. (2) holds. By (2), $S \subseteq \mathrm{sat}(S)$ for all S. Thus $\mathrm{sat}(S) \subseteq \mathrm{sat}(\mathrm{sat}(S))$. Suppose $x \in \mathrm{sat}(\mathrm{sat}(S))$. Then for each open U with $S \subseteq U$, it holds that $\mathrm{sat}(S) \subseteq U$. Thus $x \in \mathrm{sat}(S)$ by definition. Hence $\mathrm{sat}(S) = \mathrm{sat}(\mathrm{sat}(S))$, i.e., (3) holds. That (4) holds is clear. □

The following is a generalization of Theorem 4.16.

Theorem 5.15. *If F is a finite set of finite elements, then $\mathrm{sat}(F)$ is an open set of finite characteristic.*

Proof. For each $x \in F$, there is an integer n with $\mathrm{id}_n(x) = x$. Let m be the maximum of such integers. Then $\mathrm{id}_m(x) = x$ for all $x \in F$. Hence if $F \subseteq U$ for some open U, then $F \subseteq \mathrm{id}_m^{-1}(U) \subseteq U$. So $\mathrm{sat}(F) = \bigcap\{\mathrm{id}_m^{-1}(U) \mid F \subseteq U\}$. Because this is the intersection of a finite set of open sets, it is open. By the idempotence of id_m, it follows that $(\mathrm{sat}(F))^{(m)} = (\bigcap\{U^{(m)} \mid F \subseteq U, U \text{ open}\})^{(m)} = \bigcap\{(U^{(m)})^{(m)} \mid F \subseteq U, U \text{ open}\} = \bigcap\{U^{(m)} \mid F \subseteq U, U \text{ open}\} = \mathrm{sat}(F)$. □

As discussed above, in classical domain theory and topology, a set is saturated if and only if it is an upper set. But, in our setting, this relies on parallel features:

Proposition 5.16. *If every upper set is saturated, then there is*

$$(\vee)\colon \Sigma \times \Sigma \to \Sigma$$

such that $p \vee q = \top \iff p = \top$ *or* $q = \top$.

Proof. This follows directly from Theorem 5.15 and Proposition 4.20. □

In our context, a main reason for considering compact saturated sets is that definable quantifiers don't distinguish between a set and its saturation:

Proposition 5.17.

(1) Q is compact iff $\mathrm{sat}(Q)$ is compact, and in this case, $\forall_Q = \forall_{\mathrm{sat}(Q)}$.

(2) For any compact sets Q and R of the same type, it holds that $\forall_Q \sqsubseteq \forall_R$ iff $R \subseteq \mathrm{sat}(Q)$.

Proof. (1) This follows of Lemma 5.14(1). (2) $\forall_Q \sqsubseteq \forall_R$ iff $\forall U \in \mathcal{U}. \forall_Q(\chi_U) = \top \Rightarrow \forall_R(\chi_U) = \top$ iff $\forall U \in \mathcal{U}. Q \subseteq U \Rightarrow R \subseteq U$ iff $R \subseteq \bigcap\{U \in \mathcal{U} \mid Q \subseteq U\}$ iff $R \subseteq \mathrm{sat}(Q)$. □

The following is used in order to prove the main result of this subsection:

Lemma 5.18. *If Q is compact, then* $\mathrm{id}_n(Q)$ *is compact and*

$$\mathrm{id}_n(\forall_Q) = \forall_{\mathrm{id}_n(Q)}.$$

Furthermore, if U is open with $Q \subseteq U$, then there is n such that $\mathrm{id}_n(Q) \subseteq U$.

Proof. Compactness of $\mathrm{id}_n(Q)$ follows directly from Proposition 5.5(1). For each $p \in (\sigma \to \Sigma)$, we have that $\mathrm{id}_n(\forall_Q)(p) = \forall_Q(p \circ \mathrm{id}_n)$. But $\forall_Q(p \circ \mathrm{id}_n) = \top$ iff for all $x \in Q$, $p \circ \mathrm{id}_n(x) = \top$, and so $\mathrm{id}_n(\forall_Q) = \forall_{\mathrm{id}_n(Q)}$. Now if U is open with $Q \subseteq U$, then $\forall_Q(\chi_U) = \top$. Hence by rational continuity there is n such that already $\mathrm{id}_n(\forall_Q)(\chi_U) = \top$, i.e., $\forall_{\mathrm{id}_n(Q)}(\chi_U) = \top$, and so there is n such that $\mathrm{id}_n(Q) \subseteq U$. □

Theorem 5.19. *If Q is compact then* $\mathrm{sat}(Q) = \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. *Hence every compact saturated set is an intersection of upper sets of finite sets of finite elements.*

Proof. Since for any n it holds that $\mathrm{id}_n(Q) \subseteq U$ implies $Q \subseteq U$, it follows that $Q \subseteq \mathrm{sat}(\mathrm{id}_n(Q))$. Thus $\mathrm{sat}(Q) \subseteq \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. For the reverse inclusion, take any U open with $Q \subseteq U$. Then there is n such that $\mathrm{id}_n(Q) \subseteq U$ and hence $\mathrm{sat}(\mathrm{id}_n(Q)) \subseteq U$. Hence $\mathrm{sat}(Q) = \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. By Proposition 5.15, the set $\mathrm{sat}(\mathrm{id}_n(Q)$ is an open set of finite characteristic, and hence, by Proposition 4.19, it is the upper set of a finite set of finite elements. □

A family of compact sets Q_i is said to be *rationally filtered* if the chain of quantifiers $\forall_{Q_i}$ is rational in $(\sigma \to \Sigma) \to \Sigma$. In classical domain theory, algebraic domains have the property that filtered intersections of compact saturated sets are compact. This is open in our setting, even in the rational case.

Proposition 5.20. *The following are equivalent for any rationally filtered family Q_i of compact saturated subsets of a type σ:*

1. $\bigcap_i Q_i$ *is compact and* $\forall_{\bigcap_i Q_i} \sqsubseteq \bigsqcup_i \forall_{Q_i}$.
2. $\bigsqcup_i \forall_{Q_i}$ *universally quantifies over* $\bigcap_i Q_i$.
3. $\bigcap_i Q_i \subseteq U \implies \exists i. Q_i \subseteq U$ *whenever U is open.*

Proof. First observe that the reverse inequality in (1) holds by Proposition 5.17, and the reverse implication in (3) clearly holds.

(1 $\iff$ 2): Immediate from this observation.

($1 \implies 3$): The inequality (1) is equivalent to the implication

$$\forall_{\bigcap_i Q_i}(\chi_U) = \top \implies \bigsqcup_i \forall_{Q_i}(\chi_U) = \top,$$

which is clearly equivalent to $\bigcap_i Q_i \subseteq U \implies \exists i. Q_i \subseteq U$, which, in turn, is equivalent to (3).

($3 \implies 2$): We have to show that $\bigsqcup_i \forall_{Q_i}(\chi_U) = \top \iff \bigcap_i Q_i \subseteq U$. But the lhs is equivalent to $\exists i. Q_i \subseteq U$, and hence the equivalence amounts to (3) by the above observation. □

Even when the compact saturated sets Q_i are upper sets of points, say $\uparrow x_i$, we don't know whether their intersection is compact. The following proposition shows that this would be the case if x_i were a rational chain. However, it is not clear to us whether the rationality of Q_i implies that of x_i.

Proposition 5.21. *For every rational chain x_i, the intersection of the rationally filtered chain $\uparrow x_i$ of compact saturated sets is $\uparrow \bigsqcup_i x_i$ and hence is compact. Moreover, $\bigsqcup_i \forall_{\uparrow x_i} = \forall_{\uparrow \bigsqcup_i x_i}$.*

Proof. Notice that for each i, $\forall_{\uparrow x_i}(p) = p(x_i)$ and $\uparrow x_i = \mathrm{sat}\{x_i\}$, and hence $(\uparrow x_i)_i$ is a rationally filtered family of compact saturated sets. Note also that $\bigcap_i \uparrow x_i = \uparrow \bigsqcup_i x_i$ because $u \in \bigcap_i \uparrow x_i \iff \forall i. x_i \sqsubseteq u \iff \bigsqcup_i x_i \sqsubseteq u \iff u \in \uparrow \bigsqcup_i x_i$. So $\bigcap_i \uparrow x_i$ is compact. Moreover, for every $p \in (\sigma \to \Sigma)$, it holds that $(\bigsqcup_i \forall_{\uparrow x_i})(p) = \top$ iff $\forall_i p(x_i) = \top$ iff $p(\bigsqcup_i x_i) = \top$ iff $p(u) = \top$ for all $u \in \uparrow \bigsqcup_i x_i$. So $\bigsqcup_i \forall_{\uparrow x_i} = \forall_{\uparrow \bigsqcup_i x_i}$. □

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.

6.1 Program equivalence with respect to data contexts

The following is folklore and goes back to Milner [22]. We offer a proof based on the development of the previous sections:

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. In view of Proposition 4.4, it suffices to show that for any data element x of any type, $\mathrm{id}_n(x)$ is equivalent to some program with respect to ground $\mathcal{D}$-contexts. Given $x \in \sigma$ in $\mathcal{D}$, it is clear that there exist a *program* $g \in \mathtt{Baire}^m \to \sigma$ and oracles $\Omega_1, \dots, \Omega_m$ such that $x = g(\Omega_1, \dots, \Omega_m)$. It follows from m applications of Lemma 4.24 that there exist $k_1, \dots, k_m$ such that

$$\mathrm{id}_n(x) = \mathrm{id}_n(g(\mathrm{id}_{k_1}(\Omega_1), \dots, \mathrm{id}_{k_m}(\Omega_m))).$$

But the right-hand term is equivalent to a program, because clearly for any oracle Ω and n, the data term $\mathrm{id}_n(\Omega)$ is equivalent to some program. $\square$

6.2 Program totality with respect to the data language

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. $\mathtt{Cantor} \to \mathtt{Bool}$, where

$$\mathtt{Cantor} \stackrel{\mathrm{def}}{=} (\mathtt{Nat} \to \mathtt{Bool}),$$

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 [6], and can be found in [10, 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 [30]. For the intriguing relationship between totality in the Scott model with sequential computation, see [24].

6.3 Higher-type oracles

Berardi, Bezem and Coquand [7] work with a seemingly more expressive language. They have the following term-formation rule: if $t_i \colon \sigma$ is any sequence of terms, then $\lambda i.t_i \colon \mathtt{Nat} \to \sigma$ is a term. When $\sigma = \mathtt{Nat}$, this amounts to the construction of a first-order oracle, and hence we refer to the new terms as higher-type oracles. However, it turns out that the existence of such oracles follows automatically from the existence of first-order oracles:

Theorem 6.4. *In the presence of first-order oracles, for any type σ and any sequence $x_i \in \sigma$ there is $s \in (\mathtt{Nat} \to \sigma)$ such that $s(i) = x_i$ for every i.*

Proof. Any $x \in \sigma$ can be coded as a program $g \colon \mathtt{Baire}^n \to \sigma$ together with finitely many oracles $\Omega_1, \dots, \Omega_n$ such that $x = g(\Omega_1, \dots, \Omega_n)$. Using a pairing function $\langle \cdot, \cdot \rangle$, all the oracles can be packed into a single one, say Ω, and we can consider a program $h \colon \mathtt{Baire} \to \sigma$ that first unpacks the oracles and then behaves as g, so that $x = h(\Omega)$. Now, for every type τ there is an "enumerator" $E_\tau \colon \mathtt{Nat} \to \tau$ such that $E_\tau(\ulcorner t \urcorner) = t$ for any program $t \colon \tau$ with Gödel number $\ulcorner t \urcorner$. See Plotkin and Longley [20] for a purely operational proof that works with and without parallel features in the language. Hence if we define $\mathrm{ev}_\sigma(n, f) = E_{\mathtt{Baire} \to \sigma}(n)(f)$ then we get an "evaluator" $\mathrm{ev}_\sigma \colon \mathtt{Nat} \times \mathtt{Baire} \to \sigma$ such that $\mathrm{ev}_\sigma(\ulcorner h \urcorner, \Omega) = x$ for any element x coded as $h(\Omega)$ as above. To conclude, from the codings h_i, Ω_i of the given elements x_i, we form two first-order oracles $G(i) = \ulcorner h_i \urcorner$ and $A\langle i, n \rangle = \Omega_i(n)$, and then define $s(i) = \mathrm{ev}(G(i), \lambda n.A\langle i, n \rangle)$. By construction, $s(i) = x_i$, as required. $\square$

This theorem is applied in the proof of Lemma 7.11 below.

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.

7.1 Compactness of the Cantor space

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 [10, Chapter 3.11], and also follows from the more general Theorem 7.8 below, and hence we only provide the construction of the universal quantification program, with one minor improvement. We recursively define $\forall \colon (\mathtt{Cantor} \to \Sigma) \to \Sigma$ by

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

where t is some programmable total element of `Cantor`, e.g. 0^ω. The correctness proof for this program is similar to that of Theorem 5.8, 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 [10, Chapter 3.11]. Of course, the above program $\forall \colon (\mathtt{Cantor} \to \Sigma) \to \Sigma$ can still be written down. But it no longer satisfies the required specification given in Lemma 5.2(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.

7.2 The Gandy–Berger functional

The following theorem is due to Berger [8], with domain-theoretic denotational specification and proof, and it was known to Gandy, according to M. Hyland. 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*

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

such that for any total $p \in (\mathtt{Cantor} \to \mathtt{Bool})$, if $p(s) = \text{true}$ for some total $s \in \mathtt{Cantor}$, 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 $p \in (\mathtt{Cantor} \to \mathtt{Bool})$ 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, $p(\perp)$ is total and hence p is constant. □

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

Corollary 7.4. *There is a total program*

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

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

$$\forall(p) = \text{true} \iff p(s) = \text{true}\ \textit{for all total}\ s \in \mathtt{Cantor}.$$

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

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

Proof. Define a program

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

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

7.3 Simpson's functional

Simpson [33] 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. In order to avoid 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. $(s,t) \equiv_\epsilon (s',t') \Rightarrow \max(s,t) \equiv_\epsilon \max(s',t')$ *for all* $s, t, s', t' \in \texttt{Baire}$ *(total or not) and all* $\epsilon \in \mathbb{N}$*.*

Proof. It is easy to verify that the program

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

fulfills the requirements. □

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

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

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 $f \in (\texttt{Cantor} \to \texttt{Baire})$,

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

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. □

Theorems 7.3 and 7.7 rely on the compactness of the total elements of the Cantor type. Arguments similar to that of Proposition 5.6 show that these two theorems fail if the Cantor type is replaced by the Baire space.

7.4 Countable-Tychonoff functional

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 spaces is developed in [10, Theorem 13.1]. Given a sequence of universal quantifiers $\forall_{Q_i}$ for a sequence of compact sets Q_i, we wish to obtain the quantifier for the product of the compact sets.

We face two difficulties. The first is that, because our language doesn't include dependent types, we cannot assume that each compact set Q_i is contained in a different type σ_i. Hence we make the simplifying assumption that all the compact sets are contained in the same type σ. The second difficulty is that we are not able to produce a sequential algorithm without additionally being given a sequence $u_i \in Q_i$ of points. Hence we just assume that such a sequence is also given. The logically minded reader may be tempted to conjecture that the reason for this is that the Tychonoff theorem relies on the axiom of choice, and that we are avoiding the axiom by explicitly supplying a choice as input. However, using (rather weak) parallel features, an algorithm that doesn't require the choice as input is possible — see the paragraph preceding [10, Theorem 13.1]. We leave as an open problem to develop a sequential algorithm that doesn't require the choice as input.

Here is the sequential algorithm developed in [10]:

$$A \colon (\texttt{Nat} \to \sigma) \times (\texttt{Nat} \to ((\sigma \to \Sigma) \to \Sigma)) \to (((\texttt{Nat} \to \sigma) \to \Sigma)) \to \Sigma)$$
$$A(u, \alpha) = \mathrm{hd}(\alpha)(\lambda x.p(\text{if } A(\mathrm{tl}(u), \mathrm{tl}(\alpha))(\lambda s.p(x :: s)) \text{ then } u)).$$

The following was proved in [10, Section 13.1]:

Theorem 7.8. *If $Q_i \subseteq \sigma$ is a sequence of compact sets, $u_i \in Q_i$ is a sequence of points and α is a sequence such that $\alpha_i = \forall_{Q_i}$, then $\prod_i Q_i$ is compact and*

$$A(u, \alpha) = \forall_{\prod_i Q_i}.$$

Notice that Theorem 7.1 is a special case of this, with $Q_i = \{0, 1\}$, $u_i = 0$ and $\alpha(p) = p(0) \wedge p(1)$.

However, the proof of this theorem given in [10] is for the specification of the algorithm interpreted in the Scott model. As shown in [10], in the Scott model, a quantification functional $\forall_S$ is continuous if and only if the set S is topologically compact. Hence, in the above theorem, all the sets Q_i are topologically compact in the classical sense, and, thus, by the topological Tychonoff theorem, so is the product $\prod_i Q_i$. Now, compactness of the product was used in order to prove termination of the above algorithm. But, in the current setting, although the operational notion of compactness is motivated by the classical topological one, it is not the literally the same in the absence of parallel features, and hence it is not immediately clear whether the product is compact. Alex Simpson communicated to us a proof of termination of the above algorithm without assuming topological compactness of the product, establishing the operational version of Theorem 7.8 (Lemma 7.11 below). The proof that if the algorithm terminates, then it produces the correct result is essentially the same as that given in [10] (Lemma 7.10 below).

For each natural number k, define, for any u and α,

$$A^{(k)}(u, \alpha)(p) = A(u^{(k)}, \alpha^{(k)})(\lambda s.p(s^{(k)}))$$

where, for any given sequence t, we write $t_i^{(k)} = t_{i+k}$. For the remainder of this section, let Q_i, u_i and α_i be as in the premise of Theorem 7.8. We show that $A^{(k)}(u, \alpha) : ((\mathtt{Nat} \to \sigma) \to \Sigma) \to \Sigma$ is the universal quantifier of $\prod_i Q_{i+k}$. Then the theorem amounts to the special case $k = 0$.

Lemma 7.9. *For u and α as above, and any k,*

$$\begin{aligned} A^{(k)}(u, \alpha)(p) &= \alpha_k(\lambda x.p(\text{if } A^{(k+1)}(u, \alpha)(\lambda s.p(x :: s)) \text{ then } u^{(k)})) \\ &= p(\text{if } \alpha_k(\lambda x.A^{(k+1)}(u, \alpha)(\lambda s.p(x :: s)) \text{ then } u^{(k)})). \end{aligned}$$

Proof. The first equation is established by induction on k and the second by case analysis on whether $A^{(k+1)}(u, \alpha)(\lambda s.p(x :: s))$ holds for all $x \in Q_k$. □

Hence the program $B(p, k) = A^{(k)}(u, \alpha)(p)$ satisfies the equation

$$B(p, k) = p(\text{if } \alpha_k(\lambda x.B(\lambda s.p(x :: s), k + 1)) \text{ then } u^{(k)}). \tag{1}$$

Lemma 7.10. *If $B(p, k) = \top$, then $p(s) = \top$ for all $s \in \prod_i Q_{i+k}$.*

Proof. If we define

$$\begin{aligned} B_0(p, k) &= \bot \\ B_{n+1}(p, k) &= p(\text{if } \alpha_k(\lambda x.B_n(\lambda s.p(x :: s), k + 1)) \text{ then } u^{(k)}), \end{aligned}$$

then $B = \bigsqcup_n B_n$ by rational completeness. Hence if $B(p,k) = \top$ then there is an n such that $B_n(p,k) = \top$. But, by induction on n using monotonicity of p, it is clear that, for any n, the condition $B_n(p,k) = \top$ implies $p(s) = \top$ for all $s \in \prod_i Q_{i+k}$. □

As discussed above, the following proof of the converse of the previous lemma is due to Alex Simpson:

Lemma 7.11. *If* $p(s) = \top$ *for all* $s \in \prod_i Q_{i+k}$*, then* $B(p,k) = \top$.

Proof. For the sake of contradiction, assume that the premise holds but the conclusion fails, i.e. $B(p,k) = \bot$. We show by induction on j that for every j there is an element $y_{k+j} \in Q_{k+j}$ such that

$$\begin{array}{rcl} p(y_k, y_{k+1}, \ldots, y_{k+j}, \vec{\bot}) & = & \bot, \\ B(\lambda s.p(y_k :: y_{k+1} :: \cdots :: y_{k+j} :: s), k+j+1) & = & \bot. \end{array}$$

For $j = 0$, this amounts to $p(y_k, \vec{\bot}) = \bot$ and $B(\lambda s.p(y_k :: s), k+1) = \bot$. But, by Eq. (1) and the assumptions $B(p,k) = \bot$ and $p(u^{(k)}) = \top$, we must have that $p(\bot) = \bot$ and hence that $\forall x \in Q_k. B(\lambda s.p(x :: s), k+1) = \bot$, which means that such a y_k must exist. The proof of the induction step is identical, but replaces the assumption $B(p,k) = \bot$ by the induction hypothesis given by the above two equations. By Theorem 6.4, there exists $s : \mathtt{Nat} \to \sigma$ in $\mathcal{D}$ such that $s(j) = y_{k+j}$. Hence the sequences $y_k, y_{k+1}, \ldots, y_{k+j}, \vec{\bot}$ form a j-indexed rational chain with supremum s, and, by continuity, $p(s) = \bot$. However, $p(s) = \top$ because $s \in \prod_{j \geq k} Q_j$ by construction. □

We observe that this proof can be seen as a special case of that of the topological Tychonoff theorem for a well-ordered set of indices given in [38].

8 End

8.1 Remarks on parallel convergence

A function $(\vee) \in (\Sigma \times \Sigma \to \Sigma)$ such that

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

is known as *parallel convergence* or *weak parallel-or*. Abramsky showed that parallel-or on the booleans is not definable from this [1], Stoughton showed that parallel-or is equivalent to the parallel conditional at ground types [36], and Plotkin showed that the parallel conditional is not PCF-definable but that the Scott model is fully abstract for PCF extended with the parallel conditional [29]. On the other hand, it is easy to see that parallel convergence is definable from parallel-or. The Scott model of PCF fails to capture contextual equivalence, but, combining [36] and [29], it becomes fully abstract for PCF extended with parallel-or.

As we have seen, a variety of results of domain theory as applied to programming language semantics turn out to be valid in a sequential setting, despite the above mismatch of the Scott model with PCF. However, we have found that three results do depend on parallel features. But, because parallel-or is needed to obtain full abstraction, it is interesting that two of these results depend on a form of parallelism that is weaker than parallel-or:

Theorem 8.1. *The following are equivalent.*

1. *There is a parallel convergence function.*
2. *Open sets are closed under the formation of finite unions.*
3. *The upper set of any finite set of finite elements is open.*
4. *For every pair of elements $x \sqsubseteq y$ of type σ, there is a "path" $p \in (\Sigma \to \sigma)$ with $p(\bot) = x$ and $p(\top) = y$.*

Proof. The equivalence of (1)–(3) is proved in Propositions 3.7 and 4.20.

(4) $\implies$ (1): For $f, g\colon \Sigma \to \Sigma$ defined by $f(x) = x$ and $g(x) = \top$, we have $f \sqsubseteq g$, and hence a path $p\colon \Sigma \to (\Sigma \to \Sigma)$ from f to g. But then its transpose $\Sigma \times \Sigma \to \Sigma$ is parallel convergence.

(1) $\implies$ (4): This direction of the proof was communicated to us by Alex Simpson. By induction on types, define $c_\sigma\colon \Sigma \times \sigma \times \sigma \to \sigma$ by

$$\begin{aligned} c_\gamma(t, x, y) &= \text{if } t \vee x == y \text{ then } y, \\ c_{\sigma\times\sigma'}(t, \langle x, x'\rangle, \langle y, y'\rangle) &= \langle c_\sigma(t, x, y), c_{\sigma'}(t, x', y')\rangle, \\ c_{\sigma\to\tau}(t, f, g) &= \lambda x. c_\tau(t, f(x), g(x)), \end{aligned}$$

where γ is ground. Then, by induction on σ, it is easy to see that $c_\sigma(\bot, x, y)$ is the meet of x and y in the contextual order and that $c_\sigma(\top, x, y) = y$. In particular, if $x \sqsubseteq y$ then $c_\sigma(\bot, x, y) = x$. Hence we can define $p(t) = c_\sigma(t, x, y)$. □

Condition (4) hasn't shown up in our work, but it appears occasionally in synthetic and axiomatic domain theory. As far as we know, it hasn't been previously observed that this is equivalent to (1). The third aforementioned result is that if every upper set is saturated, then parallel convergence is definable (Proposition 5.16); but we don't know whether the converse holds.

From our perspective, what is interesting regarding the above theorem is that, despite the fact that the fundamental axiom of classical topology given by Theorem 8.1(2) *fails* in the absence of parallel features, a wealth of classical topological theorems on domain theory prove to be valid in a sequential setting.

8.2 Open problems and further developments

A compelling aspect of the operational development of the domain theory and topology of program types is that many of the traditional definitions arise as theorems, showing that they are inevitable. In particular, in domain-theoretic denotational semantics, one defines domains and continuous functions and then *chooses* to interpret types as domains and programs as continuous functions, motivated by intuition. Here, independently of any denotational model, *it just happens* that types *are* rationally complete orders and programs *are* continuous functions at an uninterpreted, operational level. Of course, what is relevant is the fact of experience that completeness and continuity lead to interesting applications. This is the case for both the denotational and the operational development of the theory. What is new is that, by working operationally, a wealth of domain-theoretic and topological machinery is available for *sequential* programming languages, with respect to contextual equivalence. But we have taken care of developing the theory in such a way that it also applies to languages with parallel features.

A main reason to consider new models, such as Milner's and games models, has been the fact that Scott models of sequential programming languages fail to be fully abstract. Here we have given compelling evidence, in the form of theorems and applications, that domain theory and topology are compatible with contextual equivalence of sequential programming languages, despite the failure of full abstraction of Scott models. The trick is to extract domain theory and topology from a programming language rather than to impose it via a denotational model. But the avoidance of syntactic manipulations suggests that our theory could be developed in a general axiomatic framework rather than just term models. This would make our results available to models that are not constructed from domain-theoretic or topological data, in particular games models. It is also plausible that the present development could be formalized in an operationally interpreted logic in the sense of Longley and Plotkin [20].

The main unresolved open-ended question is what class of programming languages the present theory can be developed for. Our use of sequence types of the form $(\texttt{Nat} \to \sigma)$ can be easily replaced by lazy lists by applying the bisimulation techniques of [12] to prove the correctness of evident programs that implement the SFP property for lazy lists. There is no difficulty in developing our results in a call-by-value setting. An operational domain theory of recursive types, which is built upon ideas developed here, has been developed in [15, 16] by the second-named author, where well known denotational algebraic-compactness results are established with respect to contextual equivalence. But computational features such as state, control and concurrency, and non-determinism and probability seem to pose genuine challenges. In particular, the proof of the key Lemma 4.13 doesn't go through in the presence of state or control, because extensionality fails. 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. And there are similar questions for other traditional computational effects.

Acknowledgements. The impossibility of a constructive proof of Theorem 4.16 (Remark 4.18(2)) was found together with Vincent Danos during a visit to our institution. Alex Simpson proposed the proofs of Lemma 7.11 and the implication $(1) \Rightarrow (4)$ of Theorem 8.1, and we have profited from many discussions with him. We also benefited from valuable feedback from Achim Jung, Paul B. Levy, Andy Pitts, Uday Reddy, Thomas Streicher and Steve Vickers.

References

[1] S. Abramsky. The lazy lambda-calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

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

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

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

[5] 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.

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

[7] S. Berardi, M. Bezem, and T. Coquand. On the computational content of the axiom of choice. *J. Symbolic Logic*, 63(2):600–622, 1998.

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

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

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

[11] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *Continuous Lattices and Domains*. Cambridge University Press, 2003.

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

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

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

[15] W.K. Ho. An operational domain-theoretic treatment of recursive types. In *22nd Conference on the Mathematical Foundations of Programmining Semantics*, 2006.

[16] W.K. Ho. *Operational domain theory and topology of sequential functional languages*. PhD thesis, School of Computer Science, University of Birmingham, October 2006.

[17] 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.

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

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

[20] J. Longley and G. Plotkin. Logical full abstraction and PCF. In *The Tbilisi Symposium on Logic, Language and Computation: selected papers (Gudauri, 1995)*, Stud. Logic Lang. Inform., pages 333–352. CSLI Publ., Stanford, CA.

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

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

[23] M. Mislove. Topology, domain theory and theoretical computer science. *Topology Appl.*, 89(1-2):3–59, 1998.

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

[25] D. Normann. On sequential functionals of type 3. *Math. Structures Comput. Sci.*, 16(2):279–289, 2006.

[26] 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.

[27] 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.

[28] 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.

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

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

[31] D.S. Scott. Data types as lattices. *SIAM J. Comput.*, 5:522–587, 1976.

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

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

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

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

[36] A. Stoughton. Interdefinability of parallel operations in PCF. *Theoret. Comput. Sci.*, 79(2, (Part B)):357–358, 1991.

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

[38] D.G. Wright. Tychonoff's theorem. *Proc. Amer. Math. Soc.*, 120(3):985–987, 1994.

Contents

This page is not intended to be part of a published version.