Title	Operational domain theory and topology of a sequential programming language
Author(s)	Martín Escardó and Weng Kin Ho
Source	*Information and Computation, 207*(3), 411-437
Published by	Elsevier

This document may be used for private study or research purpose only. This document or any part of it may not be duplicated and/or distributed without permission of the copyright owner.

The Singapore Copyright Act applies to the use of this document.

NOTICE: this is the author's version of a work that was accepted for publication in Information and Computation. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Information and Computation, Volume 207, Issue 3, March 2009, pages 411-437,
doi: 10.1016/j.ic.2008.12.003

Operational domain theory and topology of sequential programming languages

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

November 2006, revised May 2008

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 [34]. 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 [31]. 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 [5, 13], 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 [27] 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. [29].

Regarding topology [25, 37], 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 [4, 19], 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 [31, 12]. 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 [35]. 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 [23] and Sands (see Pitts [29] for references). Already in [23], one can find, in addition to rational-chain principles, two equivalent formulations of an operational notion of finiteness directly imported from domain theory. In addition to redeveloping their formulations in terms of rational chains rather than directed sets of terms, here we add a topological characterization, also imported from domain theory (Theorem 4.16).

The idea that topological techniques can also be directly understood in terms of operational semantics, and, moreover, are applicable to sequential languages, is due to the first-named author [12]. 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 [12] and is related to relative realizability [7] and TTE [41].

1.2 Organization

Section 2: Language, oracles, extensionality, monotonicity and rational chains.
Section 3: Rational chains, open sets and continuity principles.
Section 4: Finite elements, continuity principles and density of total elements.
Section 5: Compact sets and uniform-continuity principles.
Section 6: A data language, contextual equivalence and totality.
Section 7: Sample applications.
Section 8: Remarks on parallel convergence.
Section 9: Open problems and further work.

2 Pillars

As stated in the introduction, we never invoke evaluation mechanisms explicitly in our investigation of contextual order and equivalence. This is possible due to the existence of a large body of previous work by other authors, which we summarize here and take as our starting point. Officially, we investigate a particular "base" programming language and some extensions of a restricted form. To a large extent, in practice, what matters for our development is that, whatever language we are considering, the properties discussed in this section hold. From this point of view, our work can be considered to be axiomatic, and it may well be worthwhile to pursue this direction. However, at this stage, our aim is to develop our theory assuming an operational foundation for a restricted kind of programming language.

Different authors have defined the syntax and the operational semantics of our base language in a multitude of different, but equivalent ways. We don't wish and don't need to commit ourselves to a particular formulation. Our aim is to be mathematically rigorous but not formal, where our notion of rigour includes the requirement that the arguments are routinely formalizable when this is required.

2.1 The base 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 [31, 15, 29, 39]. Other possibilities are briefly discussed in Section 9.

2.2 Inessential, but convenient, extensions of the base language

For clarity of exposition, we explicitly include a *Sierpinski* base type $\mathcal{S}$ 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 [33]). The type $\mathcal{S}$ will have elements $\bot$ (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, without informing us that the next tick will never take place (see Section 2.8 below for a precise mathematical statement).

What is relevant for our purposes is that, for any type σ, functions $\sigma \to \mathcal{S}$ 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, $\mathcal{S}$ will classify open sets (this belongs to the realm of topology) and $\overline{\omega}$ will index rational chains (this belongs to the realm of domain theory).

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

(1) $\top : \mathcal{S}$ is a term.

(2) If $M : \mathcal{S}$ and $N : \sigma$ are terms then $(\text{if } M \text{ then } N) : \sigma$ is a term.

(3) If $M : \overline{\omega}$ is a term then $(M + 1) : \overline{\omega}$, $(M - 1) : \overline{\omega}$, and $(M > 0) : \mathcal{S}$ are terms.

Notice that there is no "else" clause in the above construction. The only value (or canonical form) of type $\mathcal{S}$ 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 $\bot_\sigma = \text{fix}\, x.x$, where fix denotes the fixed-point recursion construct. In what follows, if $f : \sigma \to \sigma$ is a closed term, we shall write $\text{fix}\, f$ as an abbreviation for $\text{fix}\, x.f(x)$.

2.3 A data language

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

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

Then the operational semantics is extended by the following rule, which generalizes the standard rules for evaluation of first-order constants:

(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, with the purpose of defining an alternative, better behaved notion of totality for programs. To emphasize that a closed term doesn't include oracles, we refer to it as a *program*.

2.4 Underlying language for Sections 3–5

We take it to be either (1) the base programming language introduced above, with the convenient extensions, (2) its extension with oracles, (3) its extension with parallel features, such as parallel-or and Plotkin's existential quantifier [31], or parallel-convergence discussed below, 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.

2.5 Full evaluation rules for the language

As discussed above, this work considers the call-by-name semantics. Apart from possibly the rules for the types $\mathcal{S}$ and $\overline{\omega}$ and for oracles, given above, the evaluation rules are well known and standard and can be found e.g. in Plotkin [31], Gunter [15], Pitts [29] or Streicher [39], among a multitude of possible references, and we omit them because we never invoke them explicitly. As mentioned in the introduction, we instead rely on extensionality, monotonicity, and rational-chain principles for contextual equivalence and order, discussed below, which follow from them [29].

2.6 Contextual equivalence and (pre)order

Recall that two terms M and N of the same type, possibly with free variables, are said to be *contextually equivalent*, here written

$$M = N,$$

if for any ground context $C[-]$, with a hole $(-)$ of the same type as M and N, that captures all the free variables of M and N, either both $C[M]$ and $C[N]$ diverge or else both evaluate to the same value. See Pitts [29] for formal details.

Similarly, M is below N in the *contextual order*, written

$$M \sqsubseteq N,$$

if for every ground context $C[-]$ as above, if $C[M]$ evaluates to a value then $C[N]$ evaluates to the same value (i.e. either $C[M]$ diverges or else both $C[M]$ and $C[N]$ converge to the same value). Clearly,

$$M = N \iff M \sqsubseteq N \wedge N \sqsubseteq M.$$

Among our four base types `Nat`, `Bool`, $\mathcal{S}$ and $\overline{\omega}$, only the first three are considered to be ground for the purpose of the above definitions. With this understanding of ground type, one has

> if M is ground and closed, then M evaluates to a value v iff $M = v$,

by considering the identity context. Moreover, it is well known that it is enough to consider the ground type $\mathcal{S}$ for the above definition [29, Remark 2.10]:

> $M \sqsubseteq N$ iff for any context $C[-]\colon \mathcal{S}$ that captures the free variables of M and N, if $C[M] = \top$ then $C[N] = \top$,

because $\top$ is the only value of type $\mathcal{S}$.

As is well-known, these two relations typically become strictly coarser (they hold less often) when the language is extended (e.g. with parallel features or effects), but we observe that they don't change in the particular case the language is extended with oracles (Section 6.2).

2.7 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 $\sigma \to \tau$. We occasionally refer to elements of function types as *functions*. With this notation, the above definitions and observations specialize to

$$x \sqsubseteq y \text{ in } \sigma \text{ iff } p(x) = \top \implies p(y) = \top \text{ for every } p \in (\sigma \to \mathcal{S}).$$

In one direction, given p one considers to context $C[-] = p(-)$, and, in the other, given a context $C[-]$, one considers the predicate $p(z) = C[z]$ (or, more formally, $p = \lambda z.C[z]$). For this argument, it is important that we are considering a call-by-name language.

2.8 The elements of $\mathcal{S}$

The elements $\bot$ and $\top$ of $\mathcal{S}$ are contextually ordered by

$$\bot \sqsubseteq \top,$$

they are contextually inequivalent, and any element of $\mathcal{S}$ is equivalent to one of them. We think of $\mathcal{S}$ as a type of outcomes of observations or semi-decisions, with $\top$ as "observable true" and $\bot$ as "unobservable false".

2.9 Classical domain theory and topology

Comments by some readers of draft versions of this paper have prompted us to clarify: when we say "classical" domain theory or topology, we don't mean domain theory or topology developed using classical logic, as opposed to intuitionistic or constructive logic, but rather domain theory and topology as they are traditionally developed, as opposed to the way they are developed here in an operational setting.

2.10 Parallel convergence

Among a number of parallel features discussed in the literature, the following turns out to play a distinguished role in showing that certain results of classical domain theory fail in a sequential operational setting (summarized in Theorem 8.1). A function

$$(\vee) \in (\mathcal{S} \times \mathcal{S} \to \mathcal{S})$$

such that for all elements $p, q \in \mathcal{S}$,

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

is known as *parallel convergence* or *weak parallel-or*. For example, such a function is definable from parallel-or or from parallel-exists, or can be introduced directly by a constant with appropriate evaluation rules.

2.11 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 $(\infty > 0) = \top$.

2.12 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 [29]. Additionally, $\bot_\sigma$ is the least element of σ.

2.13 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 $h(\mathrm{fix}\, g)$ as a least upper bound in the contextual order:

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

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

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

2.14 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 [24]. Even though full abstraction of the Scott model fails for sequential languages, proofs exploiting computational adequacy are possible [20] (see [28]). Proofs using game semantics can be found in [4, 19], and operational proofs can be found in [29, 30] (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 [23]. 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 [14] or [29] (see [18]).

2.15 Notes

As we have just seen, there are a variety of ways of establishing the operational properties we have listed. Two cases are of particular interest here. Firstly, Milner's fully abstract model of PCF has been criticized for being syntactical. However, an operationally minded reader is entitled to formulate the opposite complaint, given the amount of domain theory present in Milner's paper [24]. In truth, Milner's arguments are hybrid, and, as we shall argue in Section 4.1, they are precursors of operationally-based domain theory. Secondly, although classical domain theory doesn't give a fully abstract model of PCF, it does give a fairly explicit and applicable characterization of contextual equivalence [20, 28]. What matters here is not so much whether or not one has operational proofs of operational statements, but whether one has proofs of operational statements. For our starting point, what is relevant is that the languages under consideration have the properties stated in this section, and not how they have been proved. But there is a purely operational starting point [29], which some readers may prefer.

3 Rational chains and open sets

We begin by developing fundamental order-theoretic and topological properties of program types. As discussed in the introduction, the theory developed here has some differences with classical domain theory and topology, which arise from our desire of accommodating *sequential* programming languages.

3.1 Order

We begin by showing that rational chains turn out to coincide with internally $\overline{\omega}$-indexed chains:

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 $g(y) = y + 1$. □

The above observation is crucial for our development, and seems to be new. The novelty is slightly surprising, as both rational and $\overline{\omega}$-indexed chains have been considered for more than twenty years, in operational semantics, game semantics, and synthetic and axiomatic domain theory. In classical domain theory, $\overline{\omega}$ is instead the generic *ascending ω-chain*, and hence the above lemma explains why rational chains have a special status in our context. As discussed in the introduction, in the absence of parallel features, ascending ω-chains generally fail to have least upper bounds. Moreover, we observe in Remark 8.2 that there are (trivial) ascending ω-chains that have a least upper bound but still fail to be rational.

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, $f(\bigsqcup_n x_n) = f(l(\infty)) = l'(\infty) = \bigsqcup_n l'(n) = \bigsqcup_n f(l(n)) = \bigsqcup_n f(x_n)$. □

Rather than a proposition, the above is a definition in classical domain theory, which says what the morphisms are chosen to be. The following consequence is used in the proof of Lemma 4.9 below.

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 Proposition 3.3 to the evaluation functional $F \in ((\sigma \to \tau) \to \tau)$ defined by $F(f) = f(x)$. □

Again the situation in classical domain theory is different regarding the previous corollary. Given suitable objects for the category, e.g. Scott domains or SFP domains, in order to establish cartesian closedness one shows that the pointwise order on morphisms gives the exponential or function space. To do that, one has to show, among several other things, that the order has joins of ascending chains, and these turn out to be the pointwise joins, as in the above corollary. Here, instead, cartesian closedness for programs modulo contextual equivalence is seen to hold before one considers the notion of contextual order, because we are working with the simply typed lambda-calculus under call by name. This is used to derive the above corollary from the fact that all functions, and in particular evaluation, are rationally continuous.

3.2 Topology

In domain-theoretic denotational semantics, Sierpinski-valued continuous maps are precisely the characteristic functions of Scott open sets. More generally, in classical topology, the open sets are precisely those whose characteristic functions are continuous. We make this fact into a definition [12], relying on the fact that all programs of functional type are automatically continuous:

Definition 3.5. We say that a set U of elements of a type σ is *open* if there is $\chi_U \in (\sigma \to \mathcal{S})$ 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 $\mathcal{S}$ 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 type $(\sigma \to \mathcal{S})$. □

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 $l \in (\overline{\omega} \to (\sigma \to \mathcal{S}))$ 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.

Proposition 3.7. *The following are equivalent:*

1. *For every type, the open sets are closed under the formation of finite unions.*
2. *Parallel convergence is definable in the language.*

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 $\mathcal{S} \times \mathcal{S}$ 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 [12, Chapter 4]). The following 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 $\chi_V \in (\tau \to \mathcal{S})$ is the characteristic function of the set V then $\chi_V \circ f \in (\sigma \to \mathcal{S})$ is that of $f^{-1}(V)$. □

In classical domain theory this is typically proved by explicit manipulation of the definition of Scott topology and of continuous map, but the above argument can be applied to the classical setting. The following observation plays a crucial role in the proof of Theorem 4.16:

Lemma 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.*

Hence $\uparrow x \stackrel{\text{def}}{=} \{y \in \sigma \mid x \sqsubseteq y\} = \bigcap\{U \text{ open in } \sigma \mid x \in U\}$.

Proof. This is a reformulation of the proposition stated in Section 2.7, and the conclusion follows from the definition of intersection. □

In classical topology, the above is the definition of the *specialization order*. In classical domain theory, it is the fact that the information order coincides with the specialization order of the Scott topology. The classical domain theoretic proof relies on the fact that the lower set of any point, in the information order, is Scott closed, which may not be available in our setting, as Proposition 3.10 shows. As the above (almost tautological) proof shows, the definition of contextual order is essentially the same as the topological definition of specialization order.

Proposition 3.10. *If parallel features and oracles are not available, there are elements whose lower sets fail to be closed.*

Proof. (i) A function $\mathcal{S} \times \mathcal{S} \to \mathcal{S}$ is the characteristic function of the complement of the lower set $\{\bot_{\mathcal{S}\times\mathcal{S}}\}$ iff it is a parallel convergence function.

(ii) The characteristic function $h\colon \mathbb{N} \to \mathcal{S}$ of the Halting Set, H, exists in any of the languages under consideration. Suppose $\chi \in ((\mathbb{N} \to \mathcal{S}) \to \mathcal{S})$ is a characteristic function of the complement of the lower set of h. Then $\chi(f) = \top$ iff $f \not\sqsubseteq h$ iff there is $n \notin H$ such that $f(n) = \top$. Now, clearly there is a program $f_n \in (\mathbb{N} \to \mathcal{S})$ with a parameter n such that $f_n(m) = \top$ iff $m = n$. Define a program $c\colon \mathbb{N} \to \mathcal{S}$ by $c(n) = \chi(f_n)$. By construction, c is the characteristic function of the complement of H, which exists iff the language has oracles. □

On the other hand, if parallel-or and parallel-exists are available and the languages includes oracles, it is the case that lower sets of points are closed, simply because in this case the language is equivalent to its Scott model, by Plotkin's definability results [31]. This argument is spelled out in detail in [12] and [39].

Open sets are "rationally Scott open":

Proposition 3.11. *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., $x_n \in U$. □

Remark 3.12. Cf. Remark 6.5, which refers back to this remark. In classical domain theory, the above proposition is the definition of the Scott topology. Then one argues, *informally*, that (Scott) open sets correspond to semi-decidable properties (Smyth [37]), or observable properties (Abramsky [1, 3]), or affirmable properties (Vickers [40]). Here we have *defined* open sets to be semi-decidable sets and then *mathematically proved* that they are (rationally) Scott open. However, when the language under consideration is a data language in the sense of Sections 2.3 and 6, rather than a programming language, it makes sense to refer to semi-decidable properties as *observable properties*, reserving the terminology *semi-decidable* for the notion defined with respect to a programming language. This may, indeed, be a good mathematical articulation of the distinction between the two notions, compatible with the discussion in the above work of Abramsky's. □

4 Finite elements

We develop a number of equivalent formulations of a notion of finiteness, all of them directly imported from classical domain theory. We also give a number of technical applications, which in turn have applications to program verification, reported in Section 7.

Corollary 4.4 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, and, apart from the restriction to rational chains, is the same as in classical domain theory:

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.*

In classical domain theory, the above theorem (without the restriction to rational chains) is the *definition* of algebraic domain, and one sometimes *chooses* to use algebraic domains (of a special kind) to interpret the types of the language. Yet again, a definition of domain theory becomes a theorem in our operational setting. But it is possible to proceed in a similar way in the classical setting, as done e.g. by Streicher [39].

Remark 4.3. At this point, for the first time, the proofs will be essentially the same as the classical ones, until we reach Remark 4.13. This is good: after the foundations of operational domain theory are established, there is no essential distinction between classical and operational domain theory, regarding both the formulations of theorems and their proofs, and the notation and terminology. So, in principle, for several propositions, we could just tell the readers that their proofs are essentially the same as in classical domain theory and omit them referring them to the literature. However, there are two problems with this approach: firstly, many readers will not be acquainted with classical domain theory, and may indeed wish to use this as a bridge to approach it, and, secondly and perhaps more importantly, there is no single publication in which this set of useful properties is collected and proved without daunting mathematical detours. □

Theorem 4.2 will be proved later in this section. For the moment, we develop some consequences.

Corollary 4.4. *An element b is finite if and only if for every rational chain x_n with $b = \bigsqcup_n x_n$, there is n such that already $b = x_n$.*

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

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

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

Proposition 4.5. *$f = g$ holds in $(\sigma \to \tau)$ iff $f(b) = g(b)$ for every finite $b \in \sigma$.*

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

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

Proposition 4.6. *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 $c \sqsubseteq f(b_n)$. □

Corollary 4.7. *If U is open and $x \in U$, then there is a finite $b \sqsubseteq x$ such that already $b \in U$.*

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

In order to prove Theorem 4.2, we import the following concepts from classical domain theory (see e.g. [5]):

Definition 4.8.

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, that is, its image has finitely many equivalence classes.
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. □

The idea of SFP structure is implicit in the work of Milner [24] and was made explicit by Plotkin. The work of Milner intersects classical and operational domain theory, and can be seen as a precursor of the latter. Our constructions and proofs given below are essentially work by Milner and Plotkin, in its operational and denotational manifestations distilled in, for example, [23, 39].

Lemma 4.9. *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.*

Proof. ($\Rightarrow$): 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 $b \sqsubseteq \mathrm{id}_n(b)$.

($\Leftarrow$): 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. □

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 from this and the following lemma, which gives further information.

Definition 4.10. By a *finitary type* we mean a type that is obtained from $\mathcal{S}$ and $\mathtt{Bool}$ by finitely many applications of the product- and function-type constructions. □

Lemma 4.11. *Each type of the language is SFP.*

Moreover, 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. $\mathrm{id}_n^{\sigma\times\tau}(x,y) = (\mathrm{id}_n^\sigma(x), \mathrm{id}_n^\tau(y))$.

Proof. We construct, by induction on σ, programs

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

For the base case, we define

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

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

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

Condition (1) is easily established by induction on finitary types, and conditions (2) and (3) hold by construction.

To conclude the proof, we show that the chain $\mathrm{id}_n^\sigma \stackrel{\mathrm{def}}{=} \mathrm{d}^\sigma(n)$ is an SFP structure on σ for every type σ, 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. □

Corollary 4.12.

1. *Every element of a finitary type is finite.*

2. *If $f \in (\sigma \to \tau)$ is finite and $x \in \sigma$ is arbitrary, then $f(x) \in \tau$ is finite.*

3. *If $x \in \sigma$ and $y \in \tau$ are finite then so is $(x, y) \in (\sigma \times \tau)$.*

Proof. (1) This follows directly from Lemma 4.11(1).

(2): Pick n with $f = \mathrm{id}_n(f)$. By Lemma 4.11(2) and monotonicity, we have that $f(x) = \mathrm{id}_n(f)(x) = \mathrm{id}_n(f(\mathrm{id}_n(x))) \sqsubseteq \mathrm{id}_n(f(x))$, which shows that $f(x)$ is finite as $f(x) \sqsupseteq \mathrm{id}_n(f(x))$ by definition of deflation.

(3): Similar, using Lemma 4.11(3) instead. □

Remark 4.13. From now on, until the applications Section 7, all proofs of classical domain-theoretic and topological facts require new technical insights in our operational setting, with two exceptions clearly indicated. Cf. Remark 4.3. □

4.2 Topological characterization of finiteness

In classical domain theory, it follows directly from the definitions of finiteness and of Scott topology that an element b is finite iff its upper set $\uparrow b = \{x \mid b \sqsubseteq x\}$ is Scott open, as spelled out below. The corresponding fact also holds in our operational setting, but is less trivial. Moreover, there is a twist: to show that if b is finite then $\uparrow b$ is open amounts to showing that there is a program for the characteristic function of $\uparrow b$; we show that such a program exists, but that it cannot be explicitly exhibited in general. 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 $(\sigma \to \mathcal{S})$. □

Lemma 4.15. *For any open set U in σ and any fixed $n \in \mathbb{N}$, 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 $U^{(k)}$ is rational and $U = \bigcup_k U^{(k)}$.*

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

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

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

As mentioned above, 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 Lemma 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 3.11(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 4.7 and Proposition 3.11(1). □

Remark 4.18.

1. Notice that the proof of Theorem 4.16($\Rightarrow$) is not constructive. The reason is that we implicitly use the fact that a subset of a finite set is finite. In general, however, it is not possible to finitely enumerate the members of a subset of a finite set unless the defining property of the subset is decidable, and here it is only semi-decidable. So, although the theorem shows that the required program $\chi_{\uparrow b}$ exists, it doesn't explicitly exhibit 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 semi-decide 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-semi-decidable for finitary PCF, this would give a decision procedure for equivalence, contradicting [21]. □

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. *Parallel convergence is definable in the language.*

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 set of total elements

The results of this section are not used anywhere else in the paper, but the notion of totality, defined here, is crucial both for much of the technical development of the paper and the applications given in Section 7. We develop an operational version of the Kleene–Kreisel density theorem [11]. This is the first and only time in which we use syntactical arguments (but still without referring directly to the evaluation relation).

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 constructed 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 the term x from the terms $y_1, \dots, y_n$. □

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

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

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 $F(\infty) = \mathrm{fix}$, $F(n) \sqsubseteq G_n$ 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_n^τ. 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. Here, not only the proofs but also the formulations of the notions and theorems are new. However, all of them can be directly exported to classical domain theory with the same proofs (and could have been discovered directly within classical domain theory).

The following 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 $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(\mathrm{id}_\delta(x)))$. □

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 with non-negative real values, 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 rather than a real 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.

We don't know whether our functions are continuous in the ϵ-δ sense for all types, but we show that this is the case for 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* $\overline{\mathrm{id}}_\epsilon : \mathtt{Baire} \to \mathtt{Baire}$ *by*

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

Then $\overline{\mathrm{id}}_\epsilon(s)$ *is finite and above* $\mathrm{id}_\epsilon(s)$*, and if* $s, t \in \mathtt{Baire}$ *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* $f \in (\sigma \to \mathtt{Baire})$ *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 4.6. 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* $\gamma \in \{\mathtt{Nat}, \mathtt{Bool}\}$*,*

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

As mentioned above, we don't know whether the above continuity theorems can be generalized to other types. One of the authors would be rather surprised if they could be generalized to all types (with or without parallel features, either in our operational setting or in the classical domain-theoretic setting), but the other has a strong intuition that the generalization to all types ought to hold.

5 Compact sets

Our definition of compact is taken from [12], as are Propositions 5.5 and 5.6. However, the proof of Proposition 5.6 given in [12] relies on computability theory, whereas our proof relies on continuity, which makes it applicable to the extension of the language with oracles. All other results reported in this section are new.

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. [16]. 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 $(\sigma \to \mathcal{S})$. □

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 $(\forall_Q) \in ((\sigma \to \mathcal{S}) \to \mathcal{S})$ 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 [12]:

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*

$$N(Q, V) \stackrel{\text{def}}{=} \{f \in (\sigma \to \tau) \mid f(Q) \subseteq V\}$$

   is open in $(\sigma \to \tau)$.

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

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 $g \in (\overline{\omega} \times \mathtt{Nat} \to \mathcal{S})$ 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 $u \in (\overline{\omega} \to \mathcal{S})$ 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 $f \in (\mathtt{Baire} \to \mathtt{Nat})$ 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 iff $\{U \text{ open } \mid V \subseteq U\}$ 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 $(\mathtt{Nat} \to \sigma)$ as sequence types and define "head", "tail" and "cons" constructs for sequences as follows:

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

We also use familiar notations such as

$$0^n 1^\omega$$

as shorthands for evident terms such as

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

Theorem 5.8. *The set $\mathbb{N}_\infty$ of sequences of the forms $0^n 1^\omega$ and 0^ω is compact in the type* `Baire`.

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

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

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

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

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 4.6, 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 $\overline{\mathrm{id}}_\epsilon$: `Baire` $\to$ `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

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

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 4.6, $\forall_Q(\mathrm{id}_\delta(p)) = \top$ for some $\delta \in \mathbb{N}$, and, by Lemma 4.11(2), we have that $\mathrm{id}_\delta(p)(x) = p(\mathrm{id}_\delta(x))$. 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 $\gamma \in \{\mathtt{Nat}, \mathtt{Bool}\}$, $f \in (\sigma \to \gamma)$ 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.[35], these are sometimes referred to as the *intensional* and *extensional* moduli of continuity respectively.) □

Clearly, the small modulus of continuity is always smaller than the big one. Although they can be equal, they are different in general.

Examples 5.13. Let $f, g\colon (\mathtt{Nat} \to \mathtt{Bool}) \to \mathtt{Bool}$ be defined by $f(\alpha) = \mathrm{true}$ and by $g(\alpha) = \mathrm{if}\ \alpha_{17} == 0\ \mathrm{then\ true\ else\ true}$. Then the small and big moduli of f at $\mathbb{N}_\infty$ are both 0, but they are respectively 0 and 18 for g. □

Intuitively, the big modulus tells us how much of the input the function queries to produce the output, whereas the small one tells us how much of the argument the value of the function actually depends on.

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. The existing classical proofs don't apply to our setting, and genuinely new technical ideas are needed to establish this, but, again, the proofs offered here apply to the classical setting.

Definition 5.14. 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 5.15. *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 5.16. *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 entails the existence of parallel features:

Proposition 5.17. *If every upper set is saturated, then parallel convergence is definable in the language.*

Proof. This follows directly from Theorem 5.16 and Proposition 4.20. □

We don't know whether the converse holds. 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 5.18.

(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 5.15(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 $R \subseteq \mathrm{sat}(Q)$. □

Lemma 5.19. *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 $p \in (\sigma \to \mathcal{S})$, 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 5.20. *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 5.16, 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 $(\sigma \to \mathcal{S}) \to \mathcal{S}$. 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. We now briefly summarize what we know about this.

Proposition 5.21. *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 5.18, 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. □

Now notice that for any element x, the upper set $\uparrow x$ is compact with $\forall_{\uparrow x}(p) = p(x)$, by monotonicity of p. Even when the compact saturated sets Q_i in the above proposition are upper sets of points, say $\uparrow x_i$, we don't know whether their intersection is compact. The following proposition shows that this is the case if x_i is a rational chain. However, it is not clear to us whether the rationality of the chain $Q_i = \uparrow x_i$ of compact sets implies that of the chain of elements x_i.

Proposition 5.22. *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, being the upper set of an element, is compact. Moreover, $\bigsqcup_i \forall_{\uparrow x_i} = \forall_{\uparrow \bigsqcup_i x_i}$.*

Proof. The first part is a well-known and easy lattice-theoretic argument: $\bigcap_i \uparrow x_i = \uparrow \bigsqcup_i x_i$ because $u \in \bigcap_i \uparrow x_i \iff \forall i.x_i \sqsubseteq u \iff \bigsqcup_i x_i \sqsubseteq u \iff u \in \uparrow \bigsqcup_i x_i$. Moreover, for any $p \in (\sigma \to \mathcal{S})$, it holds that $(\bigsqcup_i \forall_{\uparrow x_i})(p) = \top$ iff $p(x_i) = \top$ for some i iff $p(\bigsqcup_i x_i) = \top$ iff $p(u) = \top$ for all $u \in \uparrow \bigsqcup_i x_i$, which shows that $\bigsqcup_i \forall_{\uparrow x_i} = \forall_{\uparrow \bigsqcup_i x_i}$. □

6 A data language

In order to obtain a more constrained and better behaved notion of totality for programs, we embed our programming language into a data language. For base types, we keep the notion of totality unchanged. But, at function types, rather than saying that a program $f \in \sigma \to \tau$ is total iff $f(x)$ is total for every $x \in \sigma$ in the programming language, we say that the program is total iff $f(x)$ is total for every $x \in \sigma$ in the data language. This definition is formulated more generally for functional data, although our primary interest is in total programs. The data language provides a notion of higher-type element that is not necessarily computable, analogous to the elements of denotational models, that functional programs can be applied to.

6.1 Operational notions of data

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 of integers 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 $\mathcal{P}$. □

As discussed above, 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.

6.2 Program equivalence with respect to data contexts

We now show that the extension of the programming language with oracles doesn't change the notion of contextual equivalence for programs. Denote by $\sqsubseteq_{\mathcal{P}}, =_{\mathcal{P}}, \sqsubseteq_{\mathcal{D}}, =_{\mathcal{D}}$ the contextual orders and equivalences of the languages $\mathcal{P}$ and $\mathcal{D}$ as defined in Section 2.6 (cf. Section 2.4). Because $\mathcal{P} \subseteq \mathcal{D}$, the first part of the following can be interpreted as saying that, for elements of $\mathcal{P}$, equivalence with respect to ground $\mathcal{P}$-contexts and equivalence with respect to ground $\mathcal{D}$-contexts coincide.

Theorem 6.2. *For all elements $x, y \in \mathcal{P}$ of the same type,*

$$x =_{\mathcal{P}} y \iff x =_{\mathcal{D}} y.$$

More generally,

$$x \sqsubseteq_{\mathcal{P}} y \iff x \sqsubseteq_{\mathcal{D}} y.$$

We rely on two lemmas (which don't depend on each other):

Lemma 6.3. *The theorem holds for ground types.*

Hence, for ground types, we shall write "=" unambiguously to denote $=_{\mathcal{P}}$ and $=_{\mathcal{D}}$.

Proof. This follows from the observation of Section 2.6 that $x = v$ iff x evaluates to v, and that, clearly, x evaluates to a ground value in $\mathcal{P}$ iff it evaluates to the same value in $\mathcal{D}$ (this last step requires a trivial proof by induction of the definition of evaluation, taking into account that, because x is a program, the rule for oracles is never invoked). □

Lemma 6.4. *In the language $\mathcal{D}$, any finite element is $\mathcal{D}$-equivalent to a program.*

Proof. By Lemmas 4.9 and 4.11 applied to the language $\mathcal{D}$ (cf. Section 2.4), every finite element of any type σ is of the form $\mathrm{id}_n(x)$ for n and $x \in \sigma$ arbitrary.

We first consider the case that $\sigma = (\mathtt{Nat} \to \mathtt{Nat})$ and that x is an oracle Ω. We construct programs f_n by induction on n:

$$f_0(k) = \bot, \qquad f_{n+1}(k) = \text{if } k == n \text{ then } n' \text{ else } f_n(k),$$

where n' denotes the natural number $\Omega(n)$, calculated at the time of defining f_{n+1}. Then clearly $f_n(k) = \mathrm{id}_n(\Omega)(k)$ for all k, and by extensionality and the fact that every non-bottom element of $\mathtt{Nat}$ is equivalent to a natural number, $f_n =_{\mathcal{D}} \mathrm{id}_n(\Omega)$.

Now, for arbitrary σ in $\mathcal{D}$ and $x \in \sigma$, it is clear that there exist a *program* $g \in \mathtt{Baire}^m \to \sigma$ and oracles $\Omega_1, \ldots, \Omega_m$ such that $x =_{\mathcal{D}} g(\Omega_1, \ldots, \Omega_m)$. It follows from m applications of Lemma 4.24 that there exist $k_1, \ldots, k_m$ such that

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

But the right-hand term is $\mathcal{D}$-equivalent to a program, because the subterms id_n and g are programs and the subterms $\mathrm{id}_{k_1}(\Omega_1), \ldots, \mathrm{id}_{k_m}(\Omega_m)$ are equivalent to programs. □

Proof of the theorem. Because there are more ground contexts in $\mathcal{D}$ than in $\mathcal{P}$, one has that $x \sqsubseteq_{\mathcal{D}} y \implies x \sqsubseteq_{\mathcal{P}} y$. To establish the converse and hence the theorem, we apply the criterion of Section 2.7 using Lemma 6.3: we assume that $p(x) = \top$ for $p \in (\sigma \to \mathcal{S})$ in $\mathcal{D}$ and show that $p(y) = \top$ too. By continuity, there is n such that $\mathrm{id}_n(p)(x) = \top$. By Lemma 6.4, there is a program $f_n =_{\mathcal{D}} \mathrm{id}_n(p)$. Then $f_n(x) = \top$ because application is a congruence. By the hypothesis that $x \sqsubseteq_{\mathcal{P}} y$ and monotonicity, $f_n(y) = \top$. Again using the fact that application is a congruence, $\mathrm{id}_n(p)(y) = \top$, and hence $p(y) = \top$ by monotonicity, as required, and the proof of the theorem is concluded. □

Remark 6.5. In the light of Remark 3.12, it makes sense to refer to contextual equivalence defined with respect to a data language as *observational equivalence*, and keep the traditional usage of the terminology *contextual equivalence* for equivalence with respect to program contexts. Then the above theorem says that observational and contextual equivalence agree. This is compatible with the fact that both terminologies are already used to refer to the same notion. The point of the theorem, using the language of Remark 3.12, is that two programs can be distinguished by observable properties iff they can be distinguished by semi-decidable properties. □

6.3 Program totality with respect to the data language

On the other hand, the notion of totality changes:

Theorem 6.6. *There are programs that are total with respect to $\mathcal{P}$ but not with respect to $\mathcal{D}$.*

This kind of phenomenon is folklore. There are programs of type e.g. $\mathtt{Cantor} \to \mathtt{Bool}$, where

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

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 [8], and can be found in [12, 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 [32]. For the intriguing relationship between totality in the Scott model with sequential computation, see [26].

6.4 Higher-type oracles

Berardi, Bezem and Coquand [9] 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 6.7. *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 [22] 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. □

This theorem is applied in the proof of Lemma 7.10 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 [12, Chapter 3.11], and also follows from the more general Theorem 7.7 below, and hence we only provide the construction of the universal quantification program, with one minor improvement. We recursively define $\forall\colon (\mathtt{Cantor} \to \mathcal{S}) \to \mathcal{S}$ 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 $\mathtt{Cantor}$, 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. □

If the data language is taken to be $\mathcal{P}$ itself, Theorem 7.1 fails for the same reason that leads to Theorem 6.6 [12, Chapter 3.11]. Of course, the program $\forall\colon (\mathtt{Cantor} \to \mathcal{S}) \to \mathcal{S}$ of the above proof 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 [10], 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 7.2. *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 $p(s) = \mathrm{true}$ 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, $p(\bot)$ is total and hence p is constant. □

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

Corollary 7.3. *There is a total program*

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

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

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

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 7.4. *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 [35] applied Corollary 7.3 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 essential 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 7.5. *There is a total program*

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

such that

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

Proof. It is easy to verify that the program

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

fulfills the requirements. □

Theorem 7.6. *There is a total program*

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

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

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

where the supremum is taken in the lexicographic order.

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

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

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

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

The base case is trivial. For the induction step, one proceeds by a further induction on the small modulus of uniform continuity of $\mathrm{hd} \circ f\colon \mathtt{Cantor} \to \mathtt{Nat}$ at the total elements of $\mathtt{Cantor}$, crucially appealing to the non-expansiveness condition given by Lemma 7.5(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 7.2 and 7.6 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 [12, 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 parallel convergence, an algorithm that doesn't require the choice as input is possible — see the paragraph preceding [12, 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 [12]:

$$\begin{aligned}A\colon (\mathtt{Nat} \to \sigma) \times (\mathtt{Nat} \to ((\sigma \to \mathcal{S}) \to \mathcal{S})) \to (((\mathtt{Nat} \to \sigma) \to \mathcal{S})) \to \mathcal{S})\\ A(u, \alpha)(p) = \mathrm{hd}(\alpha)(\lambda x.p(\mathrm{if}\, A(\mathrm{tl}(u), \mathrm{tl}(\alpha))(\lambda s.p(x :: s))\, \mathrm{then}\, u)).\end{aligned}$$

The following was proved in [12, Section 13.1]:

Theorem 7.7. *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 [12] is for the specification of the algorithm interpreted in the Scott model. As shown in [12], 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, topological 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 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 7.7 (Lemma 7.10 below). The proof that if the algorithm terminates, then it produces the correct result is essentially the same as that given in [12] (Lemma 7.9 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 7.7. We show that $A^{(k)}(u, \alpha) : ((\mathtt{Nat} \to \sigma) \to \mathcal{S}) \to \mathcal{S}$ is the universal quantifier of $\prod_i Q_{i+k}$. Then the theorem amounts to the special case $k = 0$.

Lemma 7.8. *For u and α as above, and any k,*

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

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

Proof. If we define

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

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 7.10. *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

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

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 6.7, 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 [42].

8 Remarks on parallel convergence

Abramsky showed that parallel-or on the booleans is not definable from parallel convergence [2], Stoughton showed that parallel-or is equivalent to the parallel conditional at ground types [38], 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 [31]. 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 [38] and [31], 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 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" $p \in (\mathcal{S} \to \sigma)$ 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 $f, g \colon \mathcal{S} \to \mathcal{S}$ defined by $f(x) = x$ and $g(x) = \top$, we have $f \sqsubseteq g$, and hence a path $p \colon \mathcal{S} \to (\mathcal{S} \to \mathcal{S})$ from f to g. But then its transpose $\mathcal{S} \times \mathcal{S} \to \mathcal{S}$ is parallel convergence.

(1) $\implies$ (4): This direction of the proof was communicated to us by Alex Simpson. By induction on types, define $c_\sigma \colon \mathcal{S} \times \sigma \times \sigma \to \sigma$ 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 so far, 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 5.17); but we don't know whether the converse holds. As a simple corollary of Condition (4), we have:

Remark 8.2. In the absence of parallel convergence, there are ascending ω-chains that have a least upper bound but are not rational. For example, let $x \sqsubseteq y$ in some type σ, and consider the chain that starts with x and then continues with y repeatedly. If this were indexed by $l \in (\overline{\omega} \to \sigma)$ then we could define a path from x to y by composing l with the sequential program $e \in (\mathcal{S} \to \overline{\omega})$ defined by $e(x) = \text{if } x \text{ then } 1$. □

In is an interesting question, for which we don't know the answer, whether there are ascending ω-chains which have suprema, such that for some program either the image doesn't have a supremum or if it does then it is not preserved.

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, although with significantly different proofs.

9 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 (mathematical and computational) 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 from 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 [22].

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 $(\mathtt{Nat} \to \sigma)$ can be easily replaced by lazy lists by applying the bisimulation techniques of [14] 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 [17, 18] 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 4.11 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 7.10 and of 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. *Domain Theory and the Logic of Observable Properties*. PhD thesis, University of London, Queen's College, 1987.

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

[3] S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991.

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

[5] 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.

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

[7] 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.

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

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

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

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

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

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

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

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

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

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

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

[19] 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.

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

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

[22] 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.

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

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

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

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

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

[28] 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.

[29] 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.

[30] 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.

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

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

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

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

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

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

[37] 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.

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

[39] T. Streicher. *Domain-theoretic foundations of functional programming*. World Scientific, 2006. 132pp.

[40] S. Vickers. *Topology via Logic*. CUP, 1989.

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

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

Contents