Cost-sensitive computational adequacy of higher-order recursion in synthetic domain theory

Yue Niu[a,1] Jonathan Sterling[b,2] Robert Harper[a,3]

[a] *Computer Science Department*
Carnegie Mellon University
Pittsburgh, USA

[b] *Department of Computer Science and Technology*
University of Cambridge
Cambridge, United Kingdom

Abstract

We study a cost-aware programming language for higher-order recursion dubbed $\mathbf{PCF}_{\mathsf{cost}}$ in the setting of *synthetic domain theory* (SDT). Our main contribution relates the denotational cost semantics of $\mathbf{PCF}_{\mathsf{cost}}$ to its *computational cost semantics*, a new kind of dynamic semantics for program execution that serves as a mathematically natural alternative to operational semantics in SDT. In particular we prove an internal, cost-sensitive version of Plotkin's computational adequacy theorem, giving a precise correspondence between the denotational and computational semantics for complete programs at base type. The constructions and proofs of this paper take place in the internal dependent type theory of an SDT topos extended by a *phase distinction* in the sense of Sterling and Harper. By controlling the interpretation of cost structure via the phase distinction in the denotational semantics, we show that $\mathbf{PCF}_{\mathsf{cost}}$ programs also evince a noninterference property of cost and behavior. We verify the axioms of the type theory by means of a model construction based on relative sheaf models of SDT.

Keywords: compositional cost analysis, domain theory, synthetic domain theory, type theory, PCF

Acknowledgements. This work was funded by the United States Air Force Office of Scientific Research under grants FA9550-23-1-0728 [4], MURI FA9550-21-0009 [4], FA9550-21-1-0385 [4] and the National Science Foundation under grant CCF-1901381. We thank Tristan Nguyen at AFOSR for support. Yue Niu was supported by the Air Force Research Laboratory through the NDSEG fellowship. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of AFOSR, AFRL, or NSF.

1 Introduction

In 1977 Plotkin [22] introduced a programming language for higher-order recursion, **PCF**, and defined *computational adequacy*, a fundamental notion relating the denotational and operational semantics of a programming language in terms of the computational behavior of closed programs of ground type. Since Plotkin's seminal work, computational adequacy has been developed and refined to tailor various

[1] Email: yuen@cs.cmu.edu
[2] Email: js2878@cl.cam.ac.uk
[3] Email: rwh@cs.cmu.edu
[4] Dr. Tristan Nguyen, Program Manager

programming features and is an important property in the context of denotational approaches to program verification, enabling one to bring familiar mathematical structures and equational reasoning to bear on problems regarding program behavior. One refinement takes place in both the arrangement and metatheory of the semantics. In one direction, when the ambient category in which the model is defined has sufficient logical structure, one may view the object programming language from an internal perspective, a situation that was first thoroughly developed in the context of *synthetic domain theory* [11,20]. In the other direction, one may move to a constructive metatheory such as the internal languages of generic elementary topoi or guarded type theory. The transition to these constructive, internal versions of computational adequacy has several benefits. First, by working inside topoi or type theories, one is automatically equipped with a powerful logical language to reason directly about domains and the denotational semantics, which was one of the original motivations for the development of synthetic domain theory. Second, the constructive nature of the metatheory means that one can argue that computationally adequate denotational semantics can be directly executed, as observed by De Jong [2] in his work on constructive domain theory.

Computational adequacy in cost analysis. More recently, Niu and Harper [17] identified another application of internal adequacy in the context of cost analysis in **calf** [18], a type theory in which cost is reified as a computational effect and cost analysis takes the form of equational reasoning. A natural question in this setting is the relationship between the abstract cost bounds specified and proved in **calf** and the concrete bounds derived with respect to traditional operationally-based cost models. Viewing **calf** as a semantic universe for *internal denotational semantics*, Niu and Harper [17] determined a criterion justifying the adequacy of an abstract **calf** cost model with respect to operational semantics by means of a cost-sensitive refinement of computational adequacy, which they instantiate using a variant of the Algol programming language, which featured higher-order (total) functions, first-order store and while loops.

1.1 Motivations

The aim of the present work is to extend the results of Niu and Harper [17] to *full higher-order recursion* by combining their ideas with *synthetic domain theory*, culminating in a type theory for computing with *generalized spaces* that support cost instrumentation alongside an information order for recursion.

1.1.1 Types as generalized spaces

The type theoretic philosophy is to study a real phenomenon in terms of *abstract interfaces* rendered as types; type theoretic terms, then, correspond precisely to constructions in this discourse that preserve the geometrical and topological structure at play. For example, in synthetic domain theory (SDT), types have an intrinsic topology with respect to which all functions are continuous. By adhering to an abstract interface, the type-theoretic viewpoint brings to the fore the essential logical/geometric structure of underlying objects of interest and suppresses ill-defined objects and transformations, which makes it possible to define and reason about the domain of discourse in a conceptually simple manner that is completely rigorous.

Orthogonal to the topological and order-theoretic aspects of types as spaces in synthetic domain theory, we shall impose a different kind of geometric structure called a *phase distinction*. Originating in the theory of program modules and logical relations [9,31], a phase distinction can generally refer to any situation in which types represent *indexed* structures. For example, a program module can be represented as a family of dynamic runtime components whose "shape" is determined/indexed by a static module signature; similarly, a logical relation of a type theory can be represented as a family of computability data indexed in a syntactic type. In this work, we are concerned with a similar kind of phase distinction called the *intension-extension* phase distinction, introduced by Niu *et al.* [18] to enable simultaneous reasoning about both the cost and extensional behavior of programs.

The geometric aspect of phase distinctions emerges through two operations, *restriction* and *sealing*, that manipulate types to either trivialize the fibers or the base of the indexed structure represented by a given type. In the present case, these two operations are used to implement a form of cost profiling semantics for programs that can be easily "stripped away" via restriction, enabling one to simultaneously and faithfully carry out cost analysis in one framework; we shall describe this in mathematical terms in Section 1.2.1.

In this paper we combine the intension-extension phase distinction of Niu *et al.* [18] with the domain-theoretic structure of SDT to obtain an intrinsically cost-sensitive theory of higher-order recursion in type theory. There are two main motivations for this combination. First, building on the work of Niu *et*

al. [18,17] we obtain an account of cost analysis in type theory that is compatible with general recursive programs, which we anticipate will enable more natural programming and verification techniques. Second, we will use the resulting type theory to define and study the *internal denotational semantics* of $\mathbf{PCF}_{\mathsf{cost}}$, a cost-aware version of **PCF**, and prove a cost-sensitive internal computational adequacy theorem in the sense of *op. cit.* We explain what this entails in the following section.

1.1.2 Internal denotational semantics

By *internal* denotational semantics, we mean to specify the syntax and semantics of a given programming language $\mathcal{L}$ called the *object language* as constructions internal to type theory. Typically, one may define the syntax of the object language by means of a first-order encoding in terms of (indexed) inductive data types in which we have a type $\mathsf{tp} : \mathcal{U}$ of object-level types and a type family $\mathsf{tm} : \mathsf{tp} \to \mathcal{U}$ of object-level terms. A denotational model of $\mathcal{L}$ is given by a pair of maps $[\![-]\!]_0 : \mathsf{tp} \to \mathcal{U}$ and $[\![-]\!]_1 : (A : \mathsf{tp}) \to \mathsf{tm}(A) \to [\![A]\!]$ sending types to semantic domains and terms to elements of semantic domains. Usually we will use $[\![-]\!]$ to refer to either map when the context is unambiguous. Similarly, one may define an internal operational semantics of $\mathcal{L}$ as a type family $\Downarrow : (A : \mathsf{tp}) \to \mathsf{tm}(A) \to \mathsf{tm}(A) \to \mathcal{U}$ such that the type $e \Downarrow v$ (viewed as a proposition) holds just when e evaluates to v.

Computational adequacy, first introduced in Plotkin [22], is a statement that relates the operational and denotational semantics of $\mathcal{L}$ at the type of booleans (or any other observable type): a denotational semantics is *computationally adequate* when $[\![e]\!] = b$ just when $e \Downarrow \overline{b}$, where $\overline{b}$ is the object-level boolean denoted a semantic boolean b. In the context of program verification, computational adequacy is a desirable property because the agreement of both semantics at observable types can be used to justify semantic reasoning with respect to operational behavior. In the context of cost analysis, the significance of this property is made explicit in the work of Niu and Harper [17], where the authors illustrate how a cost-sensitive refinement of internal computational adequacy can serve as the basis for validating user-defined cost models and program cost instrumentations. At a high level, results of this kind serve to ground the abstract reasoning supported by type theory to "reality" as given by operational semantics; we defer to *op. cit.* for a deeper discussion.

In this paper we will generalize the result of Niu and Harper [17] to higher-order recursion. In contrast to *op. cit.*, in this paper we prefer to work with a version of $\mathbf{PCF}_{\mathsf{cost}}$ whose execution model is given by a *computational semantics* (see Section 7). We will elaborate on the relationship between this new kind of dynamic semantics and traditional operational semantics in Section 10.

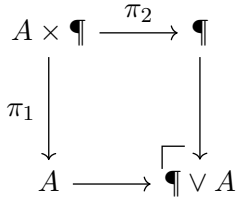
1.2 Mathematical techniques

In constructing a model for the type theory supporting both a phase distinction and recursion, we employ mathematical tools that deserve some explication, which we summarise below.

1.2.1 Cost as a phase distinction

The writer monad on a monoid is a well-established way to express the computational effect of incurring and accumulating the cost of program execution. However, as pointed out by Niu *et al.* [18], this arrangement does not faithfully model the semantics of cost profiling because it allows functions to branch on the cost component of their inputs. Thus, semantic functions lack a coherent *underlying* behavior independent of the profiling, which is indispensable for stating behavioral or correctness specifications of algorithms.

In the **calf** type theory of Niu *et al.* [18] this problem is resolved by means of the *intension-extension phase distinction*, as discussed in Section 1.1.1. Mathematically, a type-theoretic *phase* is simply a distinguished proposition $\P$ whose associated *open* and *closed* modalities [26] generate subuniverses for classifying purely extensional and purely intensional types, respectively. A *purely extensional* type A does not detect the presence of the phase proposition in the sense that the map $A \to (\P \to A)$ sending a to the constant map $\P \to A$ determined by a is an isomorphism; in other words a purely extensional type classifies only the behavior of programs. By contrast, a *purely intensional* type A "collapses" extensionally in the sense that $(\P \to A) \cong 1$. Given any type A, one may extract from A the purely extensional and purely intensional part of A by means of *idempotent monadic* modalities. The *extensional/restriction* modality is simply defined as the function space out of the proposition: $\P \to A$ (*i.e.* the reader monad on $\P$). The *intensional/sealing* modality $\P \vee -$ is defined as the following pushout/quotient inductive type (QIT) [4]:



inductive $\P \vee A : \mathsf{Set}$ **where**
$\eta_{\P \vee -} : A \to \P \vee A$
$* : \P \to \P \vee A$
$_ : (a : A) \to (u : \P) \to \eta_{\P \vee -}(a) = *(u)$

The universal property of the sealing monad can be phrased in terms of a unique extension property: for any map $A \to B$ into a purely intensional type B, there is a unique extension $\P \vee A \to B$ along the unit of the sealing monad $\eta_{\P \vee -} : A \to \P \vee A$. One can use the phase proposition $\P$ to exhibit a form of noninterference in the sense of information flow: any map $A \to (\P \to B)$ from a purely intensional type A to the extensional part of B must be constant. Therefore in **calf** one achieves a semantically faithful instrumentation of cost by programming against the writer monad on a *purely intensional* cost monoid.

From an external point of view, **calf**'s types can be modeled as presheaves on the interval $\mathbb{I} = \{\mathsf{ext} \sqsubseteq \mathsf{int}\}$; in the terminology of Sterling and Harper [32], such a presheaf encodes a set varying in the intension-extension "security poset" $\mathbb{I}$ whose presheaf restriction action *redacts* intensional cost structure. The phase proposition $\P$ is given by the "intermediate" proposition $0 \to 1$ corresponding to the representable $\mathsf{y}_{\mathbb{I}}(\mathsf{ext})$.

1.2.2 Synthetic domain theory

Synthetic domain theory (SDT) as a field started when Dana Scott conjectured that one ought to be able to reason about domains as if they were just sets *provided* that one employs a constructive ambient metalanguage. In technical terms, this metalanguage can be construed as the internal languages of a topos, *i.e.* an extensional dependent type theory. In general it is difficult to reconcile the domain-theoretic structure with the rich logical structure of dependent type theory, so the quest for SDT was in essence about constructing full subcategories of topoi that supported domain-theoretic constructions. Fullness is a critical property — it means that *every* map definable in type-theoretic language is a domain morphism, which absolves one from checking onerous side conditions when working internally. There are two well-known ways to obtain models of synthetic domain theory: one based on realizability (Hyland [11], Phoa [20], Reus [24]) and one based on sheaf topoi (Rosolini [27], Fiore and Rosolini [7], Fiore and Plotkin [5], Matache *et al.* [16]). In this paper we will construct a sheaf model of SDT, so we take a moment to recall the basic ideas, mainly drawing from the work of Fiore and Plotkin [5].

Sheaf models of SDT. The intuition behind (pre)sheaf models of synthetic domain theory is to use the Yoneda embedding to embed a category of predomains as a full subcategory of an ambient category whose internal language is dependent type theory. Thus given a (small [5]) category of predomains G, the category of presheaves on G presents the original category of predomains as a full subcategory, which constitutes a simple model of SDT. However there are some strange properties of this model from an internal perspective. For example, because the $\emptyset$ is the initial object in G, it is a sub-predomain of every other predomain. But because the embedding does not preserve colimits, the subcategory of predomains does not contain an initial object from the perspective of $\widehat{G}$; in other words, this means that while the empty set is a subset of every predomain, it is not a sub-predomain, which obstructs the use of the ambient logic to reason about (pre)domains. We may repair the model by imposing a nontrivial Grothendieck topology on G that ensures that the empty predomain is preserved by the embedding into the category of sheaves on G. In practice one may choose to preserve additional colimits as the application demands — for instance in Section 9 we will construct a model of SDT in which discrete finite predomains are preserved.

1.2.3 Predomains and the intrinsic order

Contrary to classical domain theory, ordering on predomains is a derived notion in synthetic domain theory. Nonetheless one may equip a synthetic predomain with an *intrinsic preorder* that is analogous to the information order of ordinary predomains. By default the intrinsic/synthetic preorder is not very well-behaved: it is not pointwise on limits of predomains, and it is not even a partial order in general. In this paper we shall define and base our constructions on a notion of synthetic predomains for which the

[5] The fact that natural categories of predomains are rarely small can be overcome by means of Grothendieck universes or a small dense subcategory (*e.g.* $\{\omega\} \overset{\text{full}}{\hookrightarrow} \omega\mathbf{CPO}$).

intrinsic preorder is extremely well-behaved: it is pointwise, partially ordered, and closed under *synthetic* ω-chains, which is the counterpart to closure under $\mathbb{N}$-chains for classic ω-cpos.

Aside from intrinsic interest, we are motivated by two practical incentives for considering and developing the theory of the synthetic preorder. First, we would like to connect our work to the type theory for cost analysis developed by Grodin *et al.* [8]. In that setting one works with a *cost ordering* on programs in which $e \sqsubseteq e'$ represents a proof of the fact that e' is an upper bound of e with respect to computational cost. We expect that one may extend the work of *op. cit.* to account for recursive programs by the methods we develop in this paper, and the information/intrinsic preorder appears to be the correct theoretical framework for describing the interaction of cost and partiality [13, Section 8]. Second, the advantages of the synthetic point of view outlined Section 1.2.2 do not detract from the benefits of the *language* of classical domain theory. The intrinsic preorder integrates the advantages of both the synthetic and classical perspective by providing an intuitively appealing language for reasoning about domains that automatically complies with well-definedness conditions such as monotonicity and continuity.

1.2.4 Synthetic predomains from orthogonality

From an internal perspective, predomains in a model of SDT [5,25,32] are often defined in terms of (internal) orthogonality conditions. Intuitively, an orthogonality condition can be thought of as a way of specifying a closure property with respect to a given *figure shape* $X \to Y$. A type A is *orthogonal* to a map $f : X \to Y$ when $A^f : A^Y \to A^X$ is invertible. In other words A is orthogonal to f when A "thinks" f is an isomorphism. For instance, an ω-cpo is a poset P that is orthogonal to the figure shape $\{0 \sqsubseteq 1 \sqsubseteq \dots\} \hookrightarrow \{0 \sqsubseteq 1 \sqsubseteq \dots \sqsubseteq \infty\}$, *i.e.* when P is closed under joins of $\mathbb{N}$-chains. In Section 3 we define predomains by means of multiple such orthogonality conditions. The benefit of this general approach to synthetic domain theory is that subcategories defined by (internal) orthogonality conditions are automatically (internally) reflective — this closes the predomains under structures necessary for day-to-day denotational semantics such as product and function types. Moreover, *limits* of predomains are preserved by the inclusion into the ambient SDT topos, which allows one to reason about the denotational semantics in a straightforward way using the internal language. Note that colimits of predomains always exist by virtue of the reflection, but they are usually not preserved by inclusion; in some situations this may be calibrated by ensuring a precise correspondence between the original domain-theoretic site and the subcategory of predomains (*e.g.* the internal characterization of ω-cpos of Fiore and Rosolini [6]).

1.3 Contributions

The contribution of our work is summarized as follows:

(i) An axiomatization of a type theory based on SDT that incorporates both a phase distinction and a subuniverse of predomains whose intrinsic order structure is well-behaved (Section 2).

(ii) A denotational semantics for $\mathbf{PCF}_{\mathsf{cost}}$ exhibiting noninterference between cost and behavior (Section 6).

(iii) A dynamic semantics for $\mathbf{PCF}_{\mathsf{cost}}$ in which execution is modeled directly as computation (Section 7).

(iv) An internal cost-sensitive adequacy theorem identifying the two semantics at base type (Section 8).

(v) A relative sheaf model of SDT justifying the axioms of the type theory (Section 9).

2 A type theory for cost-sensitive synthetic domain theory

We work in an extensional dependent type theory combining synthetic domain theory with the intension-extension phase distinction of Niu *et al.* [18]; the semantics and model construction of this theory is developed in Section 3 and Section 9. Here we outline the language furnished by this combination. Following Niu *et al.* [18], we assume an indeterminate proposition ¶ representing the phase distinction. We assume a (reflective) subuniverse of predomains $\mathcal{U}_{\mathsf{predom}}$. Every predomain is equipped with a *synthetic* ω-complete partial order structure, which we explain in Section 4.3. The significance of synthetic ω-completeness is that they can be used to construct the join of *rational chains* ($\mathbb{N}$-chains arising from the iterates of an endomap), the critical component in the interpretation of fixed-points in both classical and synthetic domain theory.

By a *domain* we shall mean a predomain X equipped with a least element. Equivalently this may be characterized as an algebra for the *lifting monad* $\mathbb{L} = (\mathsf{L}, \eta, \mu)$, which freely adjoins a least element to a

predomain. There is a special domain $\Sigma \cong \mathsf{L}1 \hookrightarrow \Omega$ spanned by propositions called the *Sierpiński space* or *dominance*; propositions in Σ should be thought of as the support of partial maps, and a map $X \to \Sigma$ can be thought of as a "computational" or "observable" subset of a predomain X. Moreover, because we use the *sealing modality* (see Section 1.2.1) associated with the phase proposition ¶ to define the denotational cost semantics of **PCF**, we require that predomains are closed under the sealing modality. Lastly, we assume that the subuniverse $\mathcal{U}_{\mathsf{predom}}$ is closed under lifting.

*2.1 Denotational semantics of cost-sensitive **PCF***

We aim to prove the computational adequacy property for a cost-sensitive version of **PCF** called $\mathbf{PCF}_{\mathsf{cost}}$ in which cost and partiality are treated as a single *call-by-push-value effect*. We spell out the details of the language in Section 5, but the basic idea is to separate pure values and effectful computations at both the term and type level, resulting in a class of value types A and a class of computation types X. The type structure of the language is generated by a pair of type operators F, U in which $\mathsf{F}(A)$ represents the partial cost-aware computations of type A and $\mathsf{U}(X)$ represents the value type whose underlying set of points are computations of type X. In contrast to call-by-value languages, call-by-push-value function types are *computations* and take the form $A \to X$. As an example, the ordinary call-by-value **PCF** function type $\mathsf{nat} \to \mathsf{nat}$ corresponds to the type $\mathsf{U}(\mathsf{nat} \to \mathsf{F}(\mathsf{nat}))$ in the call-by-push-value setting — here the codomain type $\mathsf{F}(\mathsf{nat})$ records both the cost incurred and the possibility of divergence, and the outer $\mathsf{U}(-)$ corresponds to the fact that functions are values in a call-by-value setting.

The denotational semantics of $\mathbf{PCF}_{\mathsf{cost}}$ is based on the adjunction models of call-by-push-value [14] in which $[\![\mathsf{F}(A)]\!]$ is defined to be the lift of $\mathbb{C} \times [\![A]\!]$ for a *purely intensional* cost monoid $\mathbb{C}$ in the sense of Section 1.2.1. By using a purely intensional type for the cost semantics, we may prove a cost-sensitive computational adequacy theorem that can be restricted to an extensional adequacy result for **PCF**. Moreover, by interpreting the cost this way, the resulting denotational cost semantics evinces a form of *information flow security* when viewing cost and behavior as security levels:

Proposition 2.1 *For any purely extensional type B, every map $f : \mathbb{C} \to \mathsf{L}B$ is* weakly constant *in the sense that for any inputs $x, y : \mathbb{C}$, we have that $f\ x$ and $f\ y$ are equal whenever they are both defined.*

As an application of computational adequacy, we can immediately transfer this intrinsic denotational security property to functions of $\mathbf{PCF}_{\mathsf{cost}}$.

Remark 2.2 As mentioned in Section 1.1.2, we will define both the syntax and semantics of $\mathbf{PCF}_{\mathsf{cost}}$ as objects and functions in the internal type theory of a synthetic domain theory topos. In a more traditional approach to denotational cost semantics, one may instead define a model of $\mathbf{PCF}_{\mathsf{cost}}$ in the category $\omega\mathbf{CPO}^{\to}$ of families of ω-cpos, which is semantically simpler in comparison to models of synthetic domain theory. In this paper we pursue an approach based on the latter in order to unify two strands of prior work — cost-sensitive programming and verification in dependent type theory [18] and internal denotational semantics [19,17] — and synthetic domain theory provides the means for smoothly integrating higher-order recursion into dependent type theory.

3 Cost-sensitive predomains in synthetic domain theory

As mentioned in the Section 1.2.3, the purpose of this section is to define a notion of predomains in synthetic domain theory such that the intrinsic preorder on predomains is partially ordered, defined pointwise on function spaces, and is closed under suprema of synthetic ω-chains. We will use these properties to define and reason about *admissible subsets of domains*. We define the basic notions of synthetic domain theory, and give an axiomatization of the rest of the paper in terms of *SDT models of the intension-extension phase distinction*, culminating in a category of predomains satisfying the properties laid out above.

3.1 Partial maps, dominance, and lifting

In a category with pullbacks, a *partial map* $A \rightharpoonup B$ is a span $A \hookleftarrow D \to B$ consisting of a subset $D \hookrightarrow A$ on which $A \rightharpoonup B$ is defined. In synthetic domain theory, only certain monomorphisms correspond to domains of definitions of partial maps. In the terminology of Rosolini [27] such a collection is called a dominion:

Definition 3.1 A pullback-stable collection of monomorphisms is a called a *dominion* when it is closed under identity and composition.

Definition 3.2 Let $\mathscr{E}$ be an elementary topos equipped with a subobject $\Sigma \hookrightarrow \Omega$ such that $\top \in \Sigma$. A monomorphism is *classified* by Σ if and only if its characteristic map factors through $\Sigma \hookrightarrow \Omega$. A *dominance* is a subobject $\Sigma \hookrightarrow \Omega$ such that the class of monos classified by Σ is a dominion. We call a proposition (resp., predicate) factoring through Σ a Σ-*proposition* (resp., Σ-*predicate*).

In the internal language, this means that Σ contains the true proposition $\top$ and is closed under dependent sums, which we write as $\phi \between f$ given $\phi : \Sigma$ and $f : \phi \to \Sigma$. These structures ensure that the dominance determines a *lifting monad* $\mathbb{L} = (\mathsf{L}, \eta, \mu)$ whose action on points are defined as follows:

Definition 3.3 The *lift* of a type A relative to a dominance Σ is defined as $\mathsf{L}A = \Sigma_{\phi:\Sigma}.\ \phi \to A$.

The lifting monad is also called the *partial map classifier* because every partial map $A \hookleftarrow D \to B$ with D a Σ-subobject of A appears as the pullback of $\eta_B : B \to \mathsf{L}B$ for a unique $A \to \mathsf{L}B$.

Notation 3.4 *Given a partial element $e : \mathsf{L}A$, we write $e{\downarrow} : \Sigma$ for its support,* i.e. *$-{\downarrow}$ is the first projection $\mathsf{L}(A) \to \Sigma$. When it is known that $e{\downarrow}$ holds, we may write $e : A$ for the defined element.*

3.2 Complete types

In synthetic domain theory the structure of predomains is generated not from consideration of $\mathbb{N}$-indexed chains but rather a new notion of chains called a *synthetic* ω-chain, which is defined simply to be a map out of ω. The difference between the two notions of chain is elucidated by the fact that $\mathbb{N}$ is the initial lifting algebra for the dominance of decidable propositions, whereas ω is the initial algebra for a larger dominance.

Definition 3.5 A *synthetic ω-chain* is a map $\omega \to A$ from the initial L-algebra or lift algebra ω.

The closure properties of synthetic ω-chains can be captured by considering an orthogonality condition relative to the figure shape $\omega \hookrightarrow \overline{\omega}$ induced by the inclusion of the initial lift algebra in the *final lift coalgebra* $\overline{\omega}$. As we alluded to in Section 1.2.4, orthogonality is a way to identify types that "think" certain maps are isomorphisms:

Definition 3.6 A type A is *orthogonal* to $f : X \to Y$ when there is a unique extension of any map $g : X \to A$ to a map $\overline{g} : Y \to A$ such that $g = \overline{g} \circ f$.

Definition 3.7 A type A is called *complete* when it is orthogonal to the figure shape $\omega \hookrightarrow \overline{\omega}$, and *well-complete* when $\mathsf{L}A$ is complete.

Complete types [11,20] are the synthetic counterpart to ω-cpos in classic domain theory. The class of well complete types was introduced in Longley and Simpson [15] as the least restrictive possible notion of predomain that is closed under lifting. In this paper we consider the dual *most restrictive* class of predomain, the *replete* types [11], in order to obtain a sharper characterisation of the intrinsic order relation (see Section 3.3) on predomains.

Definition 3.8 A map $f : X \to Y$ is called Σ-*equable* or a Σ-*isomorphism* when Σ is orthogonal to f.

Definition 3.9 A type is *replete* when it is orthogonal to every Σ-iso. A *predomain* is a replete type.

In other words, a predomain respects every Σ-isomorphism; we will use this fact to easily transfer properties that hold of Σ to every predomain in Section 4.

3.3 The intrinsic order

For every type A, the dominance Σ induces an *intrinsic preorder* $\sqsubseteq^\circ_A$ on A analogous to the specialization preorder on a topological space: $x \sqsubseteq^\circ_A y$ if and only if $f\ x$ implies $f\ y$ for every $f : A \to \Sigma$. Viewing $f : A \to \Sigma$ as a computational or observable property of A, $x \sqsubseteq^\circ_A y$ holds whenever y satisfies every observable property of x. By default the intrinsic preorder is relatively unconstrained — for instance, it need not be a partial order in general and on the dominance Σ it need not coincide with the implication

order on propositions. The purpose of this section is to axiomatize some constraints on Σ that will make intrinsic preorder coincide with the implication order on Σ; this is used in the characterization of the intrinsic order of lifting in Section 4. To this end, we introduce an intermediate *path relation* on types:

Definition 3.10 A *path* in a type A is a map $\Sigma \to A$. The *boundary* of a path $f : \Sigma \to A$ is the pair $\partial f = (f\ \bot, f\ \top)$. We write $x \sqsubseteq^{\mathsf{p}}_A y$ when there exists a path whose boundary is (x, y).

The path relation is an alternative way to surface the order structure of predomains, studied in much more detail by Fiore [3]; the path relation is also used by Grodin *et al.* [8] to obtain a theory of synthetic preorders for cost analysis. In this paper one can view the path relation as an auxiliary notion that ultimately coincides with the intrinsic preorder on predomains. In general we call types for which this holds linked, following Phoa [20]:

Definition 3.11 A type is called *linked* when the intrinsic preorder coincides with the path relation.

The fact that the intrinsic order on Σ coincides with the implication order follows if Σ is linked. This latter property holds when Σ satisfies one of the fundamental axioms of synthetic domain theory:

Definition 3.12 The dominance Σ satisfies *Phoa's principle* when the boundary evaluation map $\partial : \Sigma^\Sigma \to \Sigma \times \Sigma$ factors as an isomorphism followed by an inclusion: $\Sigma^\Sigma \cong \{(\phi, \psi) \mid \phi \to \psi\} \hookrightarrow \Sigma \times \Sigma$.

Phoa's principle expresses the fact that the path space of Σ is fully characterized by ordered pairs of Σ-propositions (with respect to implication). Because the negation of an observable property is not in general observable, Phoa's principle may be seen as an explicit statement of the constructive/observable nature of Σ-propositions, *i.e.* there is no map $\Sigma \to \Sigma$ sending ϕ to $\neg\phi$.

Proposition 3.13 *Assuming Phoa's principle, the dominance Σ is linked.*

Corollary 3.14 *Assuming Phoa's principle, the intrinsic order on Σ is the implication order.*

Moreover, the path in Σ associated to every implication $\phi \to \psi$ is unique in the following sense:

Definition 3.15 A type A is *boundary separated* when any two paths in A sharing a boundary are equal.

Proposition 3.16 *Assuming Phoa's principle, Σ is boundary separated.*

3.4 Axioms of the phase distinction in SDT and predomains

Having developed the axiomatics of the ordinary synthetic domain theoretic components of our work, we now introduce the intension-extension phase distinction as discussed in Section 1.2.1. The meeting point of the domain-theoretic and the cost-sensitive aspects of our work is simply expressed by the requirement that the phase proposition ¶ is a Σ-proposition. This allows us to manufacture a *purely intensional* monoid from a standard cost monoid that we will use to define a denotational semantics for **PCF** that exhibits the natural information flow security properties with respect to cost and behavior as outlined in Section 2.1.

More specifically, we may use the fact that ¶ is a Σ-proposition to define the sealing monad ¶ $\vee -$ mentioned in Section 1.2.1 that can be seen as the canonical way of making a predomain purely intensional. When defining the denotational semantics, we may then take as an input to the model construction any ordinary monoid (in the subuniverse of predomains) and apply the sealing monad to obtain a purely intensional monoid. Lastly, to exhibit the kind of noninterference property of cost and behavior as discussed in Section 1.2.1, we require that there is a purely extensional base predomain. These considerations lead us to the following axiomatization of our synthetic domain theory topos:

Definition 3.17 An *SDT model of the intension-extension phase distinction* is an elementary topos $\mathscr{E}$ equipped with a dominance Σ satisfying Phoa's principle such that Σ is complete, and a distinguished Σ-proposition ¶ such that the type of booleans 2 is an extensional predomain.

For the rest of the paper we assume a given SDT model of the intension-extension phase distinction $\mathscr{D}$. All constructions are all carried out in the internal language of $\mathscr{E}$.

4 Properties of predomains

We now establish the expected characterizations of the intrinsic preorder (Section 3.3) on predomains:

(i) The intrinsic preorder is pointwise on products, functions, and liftings of predomains.

(ii) The intrinsic preorder on a predomain is a *synthetic ω-complete partial order*.

(iii) The synthetic ω-complete partial order structure of predomains is defined componentwise for products and functions between predomains.

The general properties of the intrinsic preorder and link relation in SDT have been investigated in several prior works [20,24,15]. Here we recall just a few important properties that we will need.

Proposition 4.1 *Any predomain A enjoys the following properties:*

(i) Completeness*: A is orthogonal to $\omega \hookrightarrow \overline{\omega}$.*

(ii) Anti-symmetry*: the intrinsic preorder on A is a partial order.*

(iii) Boundary separation*: maps $\Sigma \to A$ with equal boundary are equal.*

(iv) Linkedness*: the intrinsic preorder and the link relation on A coincide.*

In addition, because predomains are defined in terms of orthogonality conditions, they are closed under (internal) limits and have all colimits, *i.e.* limits of predomains are computed in the same way as limits of general types. The rest of the section is dedicated to proving the desired properties on the intrinsic order on predomains. Because the intrinsic order $\sqsubseteq^\circ$ and $\sqsubseteq^{\mathsf{P}}$ are the same for predomains, we may speak of the *synthetic order* of predomains and write $\sqsubseteq$ for this relation in the rest of the paper.

4.1 Discrete predomains

We shall assume that the cost structure of $\mathbf{PCF}_{\mathsf{cost}}$ is given as a discrete type in the following sense:

Definition 4.2 A type A is called *flat* or *discrete* when $x \sqsubseteq^\circ y$ implies $x = y$.

Definition 4.3 A type *has Σ-equality* when its equality relation is valued in Σ-propositions.

Proposition 4.4 *Any type with Σ-equality is discrete.*

Proof. Let $f : A \to \Sigma$ be the characteristic map that sends a to $a = x$, which by assumption is a Σ-proposition. Since $x \sqsubseteq_A y$ on the specialization order and $f(x)$ holds, we have that $f(y)$ holds as well.□

The category of predomains possesses a natural numbers type $\mathbb{N}_{\mathsf{P}}$ with Σ-equality, which means it is also discrete. Note that it is not necessarily the same as the ambient natural numbers type $\mathbb{N}$, and we will not assume that it is the case in our constructions. From a logical perspective, this difference means that $\mathbb{N}_{\mathsf{P}}$ has a universal mapping-out property whose motive is valued in predomains rather than arbitrary types.

In Section 6 we will require the cost structure of $\mathbf{PCF}_{\mathsf{cost}}$ to be both discrete and purely intensional in the sense of Section 1.2.1; here we show that these are compatible requirements by giving sufficient conditions to obtain a purely intensional discrete type (for instance, $\P \vee \mathbb{N}$ will be discrete).

Proposition 4.5 *If A has Σ-equality, then so does $\P \vee A$.*

4.2 Characterization of the synthetic order

As mentioned at the beginning of this section, one of the primary motives of using replete types as predomains is to give a compositional characterization of the synthetic order. Roughly this means that the order relation on composite predomains can be defined in terms of the order on the constituent predomains.

Proposition 4.6 *When A and B are predomains, the synthetic orders on $A \times B$ and $A \to B$ are pointwise.*

We may also give a similar characterization of the order relation on lifted predomains:

Proposition 4.7 *Given a predomain A, we have that $x \sqsubseteq_{\mathsf{L}A} y$ if and only if $x{\downarrow}$ implies $y{\downarrow}$ and whenever $x{\downarrow}$, we have $x \sqsubseteq_A y$.*

4.3 The synthetic ω-complete partial order structure

Every predomain A contains the least upper bound of a synthetic ω-chain; we will use this fact to interpret fixed points of $\mathbf{PCF}_{\mathsf{cost}}$ in Section 6.

Proposition 4.8 *For every map $f : \omega \to A$ into a complete type A, there exists an element $a_\infty : A$ such that a_∞ is a least upper bound of f with respect to the intrinsic preorder.*

Because the intrinsic preorder on a predomain is a partial order by Proposition 4.1, we write $\bigvee f$ for the (necessarily unique) element defined in Proposition 4.8. We note that suprema of synthetic ω-chains in function spaces are computed pointwise.

4.4 Domains and admissibility

Semantically, recursive functions may be interpreted as the fixed points of endomaps of domains:

Definition 4.9 A *domain* is a predomain equipped with a L-algebra structure. Every domain D contains a least element given by postcomposing the algebra map with the unique undefined element $(\bot, !) : 1 \to \mathsf{L}D$. We write $\bot_D : D$ for the least element of D.

Proposition 4.10 *Given a map of domains $f : D \to D$, there is an element $\mathsf{fix}(f) : D$ such that $\mathsf{fix}(f)$ is the least fixed-point of f.*

Similar to classical domains, we may introduce a notion of "good subsets" of domains for which fixed-point induction is valid.

Definition 4.11 A subset of a domain D is *admissible* when it contains $\bot_D$ and is closed under suprema of synthetic ω-chains.

Proposition 4.12 (Fixed-point induction) *Given an admissible subset $A \subseteq D$ of a domain D and $f : D \to D$, to show that $\mathsf{fix}(f) \in A$ it suffices to show that $x \in A$ implies $f\ x \in A$.*

4.5 Monotonicity and continuity

As we discussed in Section 1.2.2, one of the main benefits of working in a synthetic domain theory is that maps are automatically compatible with the derived order structure:

Proposition 4.13 *Every map $f : A \to B$ between predomains is monotone in the synthetic order.*

Proposition 4.14 *Every map $f : A \to B$ between predomains is continuous in the sense that $f(\bigvee d) = \bigvee(f \circ d)$ for every synthetic ω-chain $d : \omega \to A$.*

5 $\mathbf{PCF}_{\mathsf{cost}}$: a language for cost-aware higher-order recursion

Our main technical result is the computational adequacy property for $\mathbf{PCF}_{\mathsf{cost}}$, a version of Plotkin's **PCF** [22] equipped with an abstract cost effect. The treatment of both cost structure and recursion as a call-by-push-value effect in $\mathbf{PCF}_{\mathsf{cost}}$ is inspired by [13]. The syntax of $\mathbf{PCF}_{\mathsf{cost}}$ is parameterized in a monoid $\mathbb{C}$ representing the cost structure; in Section 6 we will impose further properties on $\mathbb{C}$ when we define the denotational semantics of $\mathbf{PCF}_{\mathsf{cost}}$. As mentioned in Section 2.1, the type structure of $\mathbf{PCF}_{\mathsf{cost}}$ is generated by a pair of operators F, U that corresponds semantically to the free-forgetful adjunction between plain sets and the category of *partial cost algebras*, which are sets equipped with an action for the partial cost monad $\mathsf{L}(\mathbb{C} \times -)$. The types and terms of $\mathbf{PCF}_{\mathsf{cost}}$ are summarized in Fig. 1, defined as inductive definitions in the SDT topos $\mathscr{E}$.

6 Cost-aware denotational semantics of $\mathbf{PCF}_{\mathsf{cost}}$

Both the syntax of $\mathbf{PCF}_{\mathsf{cost}}$ and our model construction is parameterized in a monoid object $(\mathbb{C}, +, 0)$ representing the cost structure. We require the following conditions to hold:

$$
\begin{array}{ll}
\textbf{inductive } \mathsf{Ty}^+ : \mathsf{Set} \textbf{ where} & \textbf{inductive } \mathsf{Tm}^+ : \mathsf{Con} \to \mathsf{Ty}^+ \to \mathsf{Set} \textbf{ where} \\
\quad \mathsf{ans} : \mathsf{Ty}^+ & \quad \mathsf{var} : \mathsf{Var}(\Gamma, A) \to \mathsf{Tm}(\Gamma, A) \\
\quad \mathsf{nat} : \mathsf{Ty}^+ & \quad \mathsf{yes} : \mathsf{Tm}^+(\Gamma, \mathsf{ans}) \\
\quad \mathsf{U} : \mathsf{Ty}^\ominus \to \mathsf{Ty}^+ & \quad \mathsf{no} : \mathsf{Tm}^+(\Gamma, \mathsf{ans}) \\
 & \quad \mathsf{zero} : \mathsf{Tm}^+(\Gamma, \mathsf{nat}) \\
\textbf{inductive } \mathsf{Ty}^\ominus : \mathsf{tp}^+ \textbf{ where} & \quad \mathsf{succ} : \mathsf{Tm}^+(\Gamma, \mathsf{nat}) \to \mathsf{Tm}^+(\Gamma, \mathsf{nat}) \\
\quad \mathsf{F} : \mathsf{Ty}^+ \to \mathsf{Ty}^\ominus & \quad \mathsf{ap} : \mathsf{Tm}^\ominus(\Gamma, A \multimap X) \to \mathsf{Tm}^+(\Gamma, A) \to \mathsf{Tm}^\ominus(X) \\
\quad \multimap : \mathsf{Ty}^+ \to \mathsf{Ty}^\ominus \to \mathsf{Ty}^\ominus & \quad \mathsf{ret} : \mathsf{Tm}^+(\Gamma, A) \to \mathsf{Tm}^\ominus(\Gamma, \mathsf{F}(A)) \\
 & \quad \mathsf{step} : \mathbb{C} \to \mathsf{Tm}^\ominus(\Gamma, X) \to \mathsf{Tm}^\ominus(\Gamma, X) \\
\mathsf{Tm}^\ominus : \mathsf{Con} \to \mathsf{Ty}^\ominus \to \mathsf{tp}^+ & \quad \mathsf{bind} : \mathsf{Tm}^\ominus(\Gamma, \mathsf{F}(A)) \to \mathsf{Tm}^\ominus(A :: \Gamma, X) \to \mathsf{Tm}^\ominus(\Gamma, X) \\
\mathsf{Tm}^\ominus(\Gamma, X) = \mathsf{Tm}^+(\Gamma, \mathsf{U}(X)) & \quad \mathsf{ifz} : \mathsf{Tm}^+(\Gamma, \mathsf{nat}) \to \mathsf{Tm}^\ominus(\Gamma, X) \to \mathsf{Tm}^\ominus(\mathsf{nat} :: \Gamma, X) \to \mathsf{Tm}^\ominus(\Gamma, X) \\
 & \quad \mathsf{fix} : \mathsf{Tm}^\ominus(\mathsf{U}(X) :: \Gamma, X) \to \mathsf{Tm}^\ominus(\Gamma, X) \\
 & \quad \mathsf{lam} : \mathsf{Tm}^\ominus(A :: \Gamma, X) \to \mathsf{Tm}^\ominus(\Gamma, A \multimap X)
\end{array}
$$

Fig. 1. The grammar of types and terms in $\mathbf{PCF}_{\mathsf{cost}}$. We will often omit $\mathsf{Tm}^{\{+,\ominus\}}$ in the case of closed terms and write $\Gamma \vdash \{A, X\}$ for the type $\mathsf{Tm}^{\{+,\ominus\}}(\Gamma, \{A, X\})$. Given $e : \mathsf{U}\mathsf{F}A$ and $f : \mathsf{U}(A \to X)$, we also write $e; f$ for $\mathsf{bind}(e, f) : \mathsf{U}X$.

$$
\begin{array}{ll}
\tau : \mathbb{C} \times \mathsf{L}A \to \mathsf{L}(\mathbb{C} \times A) & \eta_{\mathbb{T}}(a) = \eta_{\mathsf{L}}(0, a) \\
\tau(c, (\phi, f)) = (\phi, \lambda u : \phi.\ (c, fu)) & \mu_{\mathbb{T}}(e) = (c, x) \leftarrow_{\mathbb{L}} e; (c', a) \leftarrow_{\mathbb{L}} x; (c + c', a)
\end{array}
$$

Fig. 2. Left: distributive law; right: monad structure of $\mathbb{T}$.

(i) *Computational*: $\mathbb{C}$ is a predomain with Σ-equality.

(ii) *Phase separation*: $\mathbb{C}$ is purely intensional, *i.e.* $(\P \to \mathbb{C}) \cong 1$.

That $\mathbb{C}$ is computational are used in two places in the computational adequacy proof: when reasoning about the *computational semantics* of $\mathbf{PCF}_{\mathsf{cost}}$ (see Section 7), we need $\mathbb{C}$ to be discrete (which follows from Proposition 4.4) in order to prove the property that sequential composition of computations may be decomposed (Proposition 7.4), and we need $\mathbb{C}$ to have Σ-equality when showing that the *formal approximation predicates* (see Fig. 5) associated to semantic domains are admissible. We require $\mathbb{C}$ to be purely intensional to ensure that the denotational semantics of $\mathbf{PCF}_{\mathsf{cost}}$ exhibits an *intrinsic* noninterference property of cost and behavior as sketched in Section 2.1. As discussed in Section 3.4, there is a canonical way of turning any monoid $\mathcal{M}$ into a purely intensional type by means of the sealing monad $\P \vee -$.

6.1 The partial cost monad

To model partiality and cost as a single effect, the computation types of $\mathbf{PCF}_{\mathsf{cost}}$ are interpreted as algebras for the monad $(\mathbb{T}, \eta_{\mathbb{T}}, \mu_{\mathbb{T}})$ whose action on points is defined by composing the lift and the writer monad $\mathsf{T}A = \mathsf{L}(\mathbb{C} \times A)$. The distributive law for T and the resulting monad structure is displayed in Fig. 2, where we write $x \leftarrow_{\mathbb{M}} e; f(x)$ for the induced bind operation of a monad $\mathbb{M}$ where $f : A \to \mathsf{M}(B)$ is a map into a free $\mathbb{M}$-algebra. We will also write $f^\sharp(e)$ for sequencing $e : \mathsf{M}(A)$ and $f : A \to X$ for an $\mathbb{M}$-algebra X.

6.2 The derived cost algebra

To model the cost effect $\mathsf{step} : \mathbb{C} \times X \to X$, we use the fact that every algebra for $\mathbb{T}$ is canonically an algebra for the writer monad $\mathbb{C} \times -$, which is a general property of composite monads defined from a distributive law [1, Section 2]. In our case this means that every $\mathbb{T}$-algebra has an underlying cost algebra as well; we write $\boxplus : \mathbb{C} \times X \to X$ for the cost algebra map.

Proposition 6.1 *The action of the derived cost algebra satisfies the following equations for $e : \mathsf{T}(A)$ and $f : A \to X$ for some $\mathbb{T}$-algebra X:*

$$
\begin{aligned}
c \boxplus_{\mathsf{T}(A)} e &= (e{\downarrow}, \lambda u.\ \mu_{\mathbb{C}\times -}(c, e)) \\
f^\sharp(c \boxplus_{\mathsf{T}(A)} e) &= c \boxplus_X (f^\sharp\ e) \\
(c \boxplus_{A \to X} f)\ a &= c \boxplus_X (f\ a)
\end{aligned}
$$

$$\begin{array}{ll}
[\![-]\!] : \mathsf{tp}^+ \to \mathcal{U}_{\mathsf{predom}} & [\![-]\!] : \{\Gamma, A\}(\Gamma \vdash A) \to [\![\Gamma]\!] \to [\![A]\!] \\
[\![-]\!] : \mathsf{tp}^- \to \mathsf{Alg}_{\mathbb{T}}(\mathcal{U}_{\mathsf{predom}}) & [\![-]\!] : \{\Gamma, A\}(\Gamma \vdash X) \to [\![\Gamma]\!] \to U([\![X]\!]) \\
\\
[\![\mathsf{F}A]\!] = \mathsf{T}([\![A]\!]) & [\![\mathsf{ret}(a)]\!](\gamma) = \eta_{\mathbb{T}}([\![a]\!](\gamma)) \\
[\![\mathsf{U}X]\!] = U([\![X]\!]) & [\![\mathsf{step}(c, e)]\!](\gamma) = c \boxplus [\![e]\!](\gamma) \\
[\![1]\!] = 1 & [\![\mathsf{bind}(e, f)]\!](\gamma) = \mathsf{bind}([\![e]\!](\gamma), \lambda a.\ [\![f]\!](a, \gamma)) \\
[\![\mathsf{ans}]\!] = 2 & [\![\mathsf{fix}(f)]\!](\gamma) = \mathsf{fix}(\lambda x.\ [\![f]\!](x, \gamma)) \\
[\![\mathsf{nat}]\!] = \mathbb{N}_{\mathsf{P}} & \\
[\![A \rightharpoonup X]\!] = [\![A]\!] \to [\![X]\!] &
\end{array}$$

Fig. 3. Selected clauses of the model; in the above we write $U(X)$ for the carrier of a $\mathbb{T}$-algebra X.

$$\frac{}{\mathsf{bind}(\mathsf{ret}(a), f) \mapsto 0, f(a)} \qquad \frac{}{\mathsf{ap}(\mathsf{lam}(e), e_1) \mapsto 0, e(e_1)} \qquad \frac{}{\mathsf{fix}(e) \mapsto 0, e(\mathsf{fix}(e))} \qquad \frac{}{\mathsf{ifz}(\mathsf{zero}, e_0, e_1) \mapsto 0, e_0}$$

$$\frac{}{\mathsf{ifz}(\mathsf{succ}(v), e_0, e_1) \mapsto 0, e_1(v)} \qquad \frac{}{\mathsf{step}^c(e) \mapsto c, e}$$

Fig. 4. Rules for the small-step transition relation.

6.3 Denotational semantics of $\boldsymbol{PCF}_{\mathsf{cost}}$

The semantics of $\mathbf{PCF}_{\mathsf{cost}}$ is based around the free-forgetful adjunction associated with the partial cost monad in which value types are predomains and computation types are $\mathbb{T}$-algebra valued in predomains. The essential parts of the model is displayed in Fig. 3. Most of the type structure of $\mathbf{PCF}_{\mathsf{cost}}$ is interpreted using the cartesian closed structure of predomains; note that the numerals type is interpreted as the natural numbers type of predomains $\mathbb{N}_{\mathsf{P}}$, which is not the same as the ambient natural numbers $\mathbb{N}$.

7 Computational semantics of $\mathbf{PCF}_{\mathsf{cost}}$

Computational adequacy is a property relating the denotational semantics of a language with its execution behavior as given by an *operational semantics*. Commonly one employs a *structural operational semantics* [21] in which the operational semantics of a language is defined as an inductive family of relations $\mapsto_A \subseteq A \times A$. Both termination and evaluation can be defined by means of the reflexive transitive closure of the family of relations $\mapsto_A$, which is the smallest reflexive, transitive family of relations containing $\mapsto_A$.

Computational semantics. We consider an alternative formulation of the dynamic semantics of $\mathbf{PCF}_{\mathsf{cost}}$ in which execution is modeled not as an inductive family but directly as a (partial) computation. This departure from traditional operational semantics is necessitated by our proof of computational adequacy. As we explain in Section 8.2, termination must be a predicate valued in Σ-propositions; however, the reflexive-transitive closure of a decidable relation (such as $\mapsto_A$) can only be seen to be a Σ-predicate assuming that Σ is closed under countable joins of decidable propositions that are *preserved* by the inclusion into the ambient type theory, *i.e.* $\exists[n : \mathbb{N}]\ \phi(n) \in \Sigma$ for every countable family of Σ-propositions ϕ. Our approach bypasses this question by de-emphasizing $\mathbb{N}$ in favor of the initial lifting algebra ω, a lesson of Simpson [28]. We discuss the relationship between this computational semantics and ordinary operational semantics in Section 11.

7.1 Computational semantics of $\boldsymbol{PCF}_{\mathsf{cost}}$

We begin with a family of small-step transition relations $\mapsto_A \subseteq \mathsf{Tm}^+(A) \times \mathbb{C} \times \mathsf{Tm}^+(A)$ that implements the *cost effect* model in the sense of Hoffmann [10]. The intuitive meaning of $e \mapsto c, e'$ is that e transitions in one step to e' and incurs cost c; the only place where cost is effected is at step: $\mathsf{step}^c(e) \mapsto c, e$. The family of relations $\mapsto_A$ is defined as an inductive family whose generators are displayed in Fig. 4 (the expected congruence rules have been omitted). We iterate the "one step" relation to obtain the partial map implementing the computational semantics. Because $\mapsto_A$ is decidable, we have a characteristic map

$\mathsf{out} : \{A : \mathsf{tp}^+\}.\ \mathsf{Tm}^+(A) \to 1 + (\mathbb{C} \times \mathsf{Tm}^+(A))$. Consider the following functional:

$$\Phi_{\mathsf{eval}}\ f\ (e, v) = \begin{cases} c \boxplus f(e', v) & \mathsf{out}(e) = \mathsf{inr} \cdot (c, e') \\ (e = v, \lambda - .\ 0) & \mathsf{out}(e) = \mathsf{inl} \cdot \star \end{cases}$$

Define $\mathsf{eval} : \{A : \mathsf{tp}^+\}.\ \mathsf{Tm}^+(A) \times \mathsf{Tm}^+(A) \to \mathsf{T}(1)$ to be the fixed-point of Φ_{eval} and $\mathsf{profile} : \mathsf{tm}^+(\mathsf{UF1}) \to \mathsf{T}(1)$ as $\mathsf{profile}(e) = \mathsf{eval}(e, \mathsf{ret}(\star))$. The meaning of $\mathsf{eval}_A(e, v) : \mathsf{T}(1)$ is that when it is defined, e computes to a value v incurring the defined cost. Similarly, $\mathsf{profile}(e)$ is the cost of computing e when the former is defined. In the following, we will establish some expected properties of the computational semantics such as the uniqueness of evaluation and a "big-step" law for sequencing evaluations.

Proposition 7.1 *The relation* $\pi_1 \circ \mathsf{eval}$ *is functional,* i.e. $\downarrow\mathsf{eval}(e, v)$ *and* $\downarrow\mathsf{eval}(e, v')$ *implies* $v = v'$.

Proposition 7.2 *If* $\mathsf{eval}(e, \mathsf{ret}(v)) = c_1$ *and* $\mathsf{eval}(g\ v, \mathsf{ret}(w)) = c_2$, *then* $\mathsf{eval}(e; g, \mathsf{ret}(w)) = c_1 + c_2$.

Corollary 7.3 *If* $\mathsf{eval}(e, \mathsf{ret}(v)) = c_1$ *and* $\mathsf{profile}(g\ v) = c_2$, *then* $\mathsf{profile}(e; g) = c_1 + c_2$.

In ordinary operational semantics, one has the property that $e; g \Downarrow v$ implies there exists a value w such that $e \Downarrow w$ and $g(w) \Downarrow v$. An analogous rule also holds for the computational semantics, except the value w is not existentially quantified:

Proposition 7.4 *The following inference rule is valid for any* Σ*-predicate* φ*:*

$$\frac{\forall[v : A]\ \mathsf{eval}(e, v)\downarrow \wedge\, \mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(e, v) + \mathsf{eval}(g\ v, \mathsf{ret}(w)))}{\mathsf{eval}(e; g, \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(e; g, \mathsf{ret}(w)))}$$

Using this rule we may derive the following law for profiling compound sequences:

Proposition 7.5 *We have* $\mathsf{profile}((e; g); i) = \mathsf{profile}(e; (\lambda v.\ g\ v; i))$.

8 Computational adequacy and noninterference

In this section we show that the computational and denotational semantics satisfy a tight correspondence at the type of *observations*: for every $e : \mathsf{UF1}$, we have that $[\![e]\!]$ is Kleene equivalent to $\mathsf{profile}(e)$ in the sense that the cost specified computationally and denotationally are equal whenever one of them is defined.

8.1 Soundness

In one direction, it is not too difficult to show *soundness*, which means that the computational steps are respected by the denotational semantics:

Proposition 8.1 *If* $e \mapsto c, e'$, *then* $[\![e]\!] = c \boxplus [\![e']\!]$.

Theorem 8.2 *If* $\mathsf{eval}(e, v)\downarrow$, *then* $[\![e]\!] = \mathsf{eval}(e, v) \boxplus [\![v]\!]$.

Corollary 8.3 *Given* $e : \mathsf{UF1}$, *we have* $\mathsf{profile}(e) \sqsubseteq [\![e]\!]$.

8.2 Adequacy

Adequacy proper usually refers to the converse direction of the property stated in Corollary 8.3: definedness of the denotational semantics implies termination under the computational semantics. Our proof is based on a standard binary logical relations construction between the syntax and semantics of $\mathbf{PCF}_{\mathsf{cost}}$ (*cf.* Plotkin [22]). The logical relation consists of a family of relations $\lhd_A \subseteq [\![A]\!] \times \mathsf{Tm}^+(A)$ indexed in the syntactic types A of $\mathbf{PCF}_{\mathsf{cost}}$ such that $[\![e]\!] \lhd_{\mathsf{UF1}} e$ implies the computational adequacy property. The purpose of considering a family of relations is to provide a sufficient strengthening of the desired property to all types so that one may proceed by an inductive proof on the derivation of terms to show that $[\![e]\!] \lhd_A e$ holds for every term $e : A$. Due to the presence of fixed-point computations, we must show that $- \lhd_{\mathsf{U}X} e$

$$
\begin{aligned}
e \lhd_{\mathsf{1}} e' &= \top \\
e \lhd_{\mathsf{ans}} e' &= (\overline{e} = e') \\
e \lhd_{\mathsf{nat}} e' &= (e = [\![e']\!]) \\
e \lhd_{\mathsf{U}\mathsf{F}A} e' &= \forall[f\ (\lhd_A \Rightarrow \mathsf{adq})\ f']\ \mathsf{adq}(f^\sharp(e), e'; f') \\
e \lhd_{\mathsf{U}(A \to X)} e' &= e\ (\lhd_A \Rightarrow \lhd_{\mathsf{U}X})\ e'
\end{aligned}
\qquad
\begin{aligned}
\mathsf{adq}(e, e') &= (e \sqsubseteq e') \\
e\ (R \Rightarrow S)\ e' &= \forall[a\ R\ a']\ (e\ a)\ S\ (e'\ a')
\end{aligned}
$$

Fig. 5. *Formal approximation relations.* We write $\overline{-} : 2 \to \mathsf{ans}$ for the function sending 0 to no and 1 to yes. The relation $\mathsf{adq} \subseteq \mathsf{T}(1) \times \mathsf{Tm}^+(\mathsf{UF1})$ is the "ground relation" that generates the formal approximation relations at higher types.

is always an admissible subset of the domain $[\![X]\!]$ in the sense of Definition 4.11. We define a family of relations called the *formal approximation relations* in Fig. 5 by induction on the structure of syntactic types and show that they satisfy the properties in the preceding discussion.

Formal approximation relations may be extended to open terms as usual. We write $\Gamma \vdash e \lhd_A e'$ when for all closing substitutions $\sigma : \Gamma$, we have that $e([\![\sigma]\!]) \lhd_A e'[\sigma]$ holds. The computational adequacy result may be deduced from the *fundamental lemma*:

Theorem 8.4 *For every closed term $e : \Gamma \vdash A$, the approximation $\Gamma \vdash [\![e]\!] \lhd_A e$ holds.*

The proof of the fundamental lemma proceeds by induction on the derivation of terms. Details and the proof that formal approximation predicates $- \lhd_{\mathsf{U}X} e$ are admissible can be found in Appendix F; crucially we rely on the fact that $\mathsf{eval}(e, v)$ is a Σ-proposition.

Corollary 8.5 *Given $e : \mathsf{UF1}$, we have that $[\![e]\!] = \mathsf{profile}(e)$.*

Extensionally, both the denotational and computational semantics of e are simply partial computations of type $\mathsf{L1}$, so one may view Corollary 8.5 as a cost-sensitive (and internal) version of Ploktin's original adequacy theorem for **PCF**. Lastly, we see that our semantics of $\mathbf{PCF}_{\mathsf{cost}}$ provides a rigorous proof of the intuitive fact that computations may not observe the cost effect:

Theorem 8.6 *Any $e : \mathsf{UF1} \rightharpoonup \mathsf{F2}$ is weakly, extensionally constant in the sense that for all $x, y : \mathsf{UF1}$, if $\mathsf{profile}(x)\!\downarrow$ and $\mathsf{profile}(y)\!\downarrow$, then $\mathsf{eval}(e\ x, \mathsf{ret}(v))\!\downarrow$ and $\mathsf{eval}(e\ y, \mathsf{ret}(u))\!\downarrow$ imply $v = u$.*

9 An SDT model of the intension-extension phase distinction

To obtain a model for the constructions of the preceding sections, we instantiate the sheaf model of Sterling and Harper [32] at the poset $\mathbb{I} = \{\mathsf{ext} \sqsubseteq \mathsf{int}\}$ representing the intension-extension security order. The basic idea is to first develop domain theory *internal* to the presheaf topos $\widehat{\mathbb{I}}$, from which we may obtain an appropriate *internal* domain-theoretic site that embeds into a sheaf topos model of SDT in the sense of Definition 3.17. The reason to consider internal sites is that we may build into the base category the intension-extension phase distinction that is preserved through the embedding.

In *op. cit.* the internal domain theory of $\widehat{\mathbb{I}}$ is developed in terms of constructive dcpos following the work de Jong [2]. These internal dcpos are similar to ordinary dcpos; for example, one may use them to give the denotational semantics of **PCF** [2]. The difference lies in the dominance Σ of the category of internal dcpos given by the subobject classifier $\Omega_{\widehat{\mathbb{I}}}$: while a partial element of ordinary domains in **Set** is either defined or not, a partial element of a domain internal to $\widehat{\mathbb{I}}$ may have the phase proposition ¶ as its support, where ¶ is the intermediate proposition in $\Omega_{\widehat{\mathbb{I}}}$. Recalling the interpretation of a map $A \to \Sigma$ as a computational predicate, this also means that predicates can be phase-dependent in the sense of holding only at the extensional phase.

We recall some definitions from Sterling and Harper [32] (SH22).

Definition 9.1 A *Scott-open immersion* of a dcpo is any mono $U \rightarrowtail A$ arising from a predicate $A \to \Sigma$.

Definition 9.2 In a category, a *sink* on an object A is a set of morphisms into A.

Definition 9.3 In a category with pullbacks, a *Cartesian coverage* is an assignment of objects A to set of sinks on A that is stable under pullback.

Definition 9.4 The *finite open cover topology* is generated by the Cartesian coverage assigning to each dcpo A the set of sinks $\{U_i \rightarrowtail A\}_i$ on A with every $U_i \rightarrowtail A$ a Scott-open immersion and $\bigvee_i U_i \cong A$.

Our domain-theoretic site is given by an internal category $\mathscr{C}$ of small dcpos in $\widehat{\mathbb{I}}$. We embed $\mathscr{C}$ into a Grothendieck topos $\mathrm{Sh}(\mathscr{C})$, obtained by localizing $\widehat{\mathscr{C}}$ at the finite open cover topology. The purpose of this localization is to ensure that the finite joins of the dominance in $\mathscr{C}$ are preserved by the embedding into $\mathrm{Sh}(\mathscr{C})$. This property was notably used by *op. cit.* to implement the semantics of termination declassification; here we use the finite join structure of Σ to show that 2 is an extensional predomain. The phase distinction in $\mathrm{Sh}(\mathscr{C})$ is inherited from the ambient presheaf topos, where it is represented by ext.

Theorem 9.5 (SH22, Corollary 90) *Every representable presheaf is well-complete.*

Thus we obtain a functor $y : \mathscr{C} \hookrightarrow \mathrm{Sh}(\mathscr{C})$ restricting the Yoneda embedding $\mathsf{y}_{\mathscr{C}} : \mathscr{C} \hookrightarrow \widehat{\mathscr{C}}$ onto sheaves.

Theorem 9.6 (SH22, B.1.4.3) *The representable $\Sigma = y(\Sigma)$ is a dominance in $\mathrm{Sh}(\mathscr{C})$.*

Theorem 9.7 (SH22, Corollary 79) *Coproducts in $\mathscr{C}$ are given by unions of families of Scott-opens.*

Corollary 9.8 *Finite coproducts of $\mathscr{C}$ are preserved by the embedding into sheaves.*

Theorem 9.9 (SH22, Axiom-SDT-1) *The dominance Σ has finite joins that are preserved by the inclusion $\Sigma \hookrightarrow \Omega$.*

Theorem 9.10 *Setting $\Sigma = y(\Omega_{\widehat{\mathbb{I}}})$ and $\P = y(\mathsf{y}_{\mathbb{I}}(\mathsf{ext}))$, we have that $(\mathrm{Sh}(\mathscr{C}), \Sigma, \P)$ is an SDT model of the intension-extension phase distinction in the sense of Definition 3.17.*

Proof. By Theorem 9.5 we know that Σ is a well-complete dominance in $\mathrm{Sh}(\mathscr{C})$. To show that $\mathrm{Sh}(\mathscr{C})$ also models the intension-extension phase distinction, we observe that the presheaf model of the intension-extension phase distinction of Niu *et al.* [18] restricts to a smaller model in $\mathscr{C}$: every subterminal object in $\widehat{\mathbb{I}}$ is an internal dcpo, and so we may take the subterminal $\P = \mathsf{y}_{\mathbb{I}}(\mathsf{ext})$ to be the phase proposition in $\mathscr{C}$. The phase proposition $\P = y(\P)$ in $\mathrm{Sh}(\mathscr{C})$ is classified by Σ: since y is fully faithful, every $\phi : \Sigma$ arise from a unique map $1_{\widehat{\mathbb{I}}} \to \Omega_{\widehat{\mathbb{I}}}$.

Moreover, we can directly verify that $2_{\mathscr{C}} = 2_{\widehat{\mathbb{I}}} = 1_{\widehat{\mathbb{I}}} + 1_{\widehat{\mathbb{I}}}$ is internally orthogonal to $\P$ in $\mathscr{C}$. By Theorem 9.8, y preserves finite coproducts, so we have that $2_{\mathrm{Sh}(\mathscr{C})} = y(2_{\mathscr{C}})$. We observe that 2 is extensional because the restricted embedding $y : \mathscr{C} \to \mathrm{Sh}(\mathscr{C})$ is both full and faithful and preserves products. To see that 2 is replete, we observe that 2 is isomorphic to its type of singletons:

$$2 \cong \left\{\phi : \Sigma^2 \mid (\forall [a, b : 2]\ \phi\ a \wedge \phi\ b \to a = b) \wedge (\phi(\mathsf{inl} \cdot \star) \vee \phi(\mathsf{inr} \cdot \star))\right\}$$

Because Σ is closed under finite joins (by Theorem 9.9), the type of singletons of 2 can be defined as the limit of a diagram of replete types, and so it is replete as well.

Lastly, to see that Phoa's principle is satisfied, we note that it holds in $\mathscr{C}$. Since the subobject $\{(\phi, \psi) : \Sigma \times \Sigma \mid \phi \to \psi\}$ can be defined as the equalizer of $\pi_1 : \Sigma \times \Sigma \to \Sigma$ and $\wedge : \Sigma \times \Sigma \to \Sigma$, every object in the diagram is defined using the cartesian closed structure of $\mathscr{C}$, it is preserved by any cartesian closed embedding, and so Phoa's principle also holds in $\mathrm{Sh}(\mathscr{C})$. □

10 Discussion of related work

Cost analysis in type theory. The original motivation for proving internal, cost-sensitive computational adequacy results grew out of the work of Niu *et al.* [18] on formalizing cost analysis of functional programs in dependent type theory. Niu and Harper [17] prove such an adequacy theorem for a variant of the Algol language featuring a notion of first-order recursion in the form of while loops. The purpose of the present paper is to generalize that result to account for higher-order recursion. In contrast to both prior works, we have de-emphasized the role of the call-by-push-value language and instead work directly with the internal language of the SDT topos. Because the model we construct in Section 9 can also be seen as a model for the type theories in both prior works, we expect that it would be routine to formalize our results in a call-by-push-value version of type theory as well.

Cost-sensitive computational adequacy. The kind of cost-sensitive adequacy theorem we prove in this paper has been proved in a classic domain-theoretic setting by Kavvos *et al.* [13], and the general theory of computational adequacy for languages with algebraic effects has been developed by Plotkin and Power [23]. The main difference between our work and those mentioned is that we aim to prove adequacy results internally to a type theory equipped with a phase distinction (such as the one in Niu *et al.* [18]). As argued in Niu and Harper [17], such internal adequacy theorems can be used as a basis for the validity of axiomatic cost analysis in these type theories.

Relative sheaf models of SDT. As we have explained in Section 1.2.2, the main sheaf models of synthetic domain theory take the form of Grothendieck topoi over the category of sets. Meanwhile, the logic of phase distinctions finds its home in presheaf topoi in which one finds many distinct subterminal objects that are neither globally true nor globally false; because the category of sets is boolean and two-valued, it can have no non-trivial phase distinctions. For this reason, Sterling and Harper [32] have proposed to combine synthetic domain theory with phase distinctions by developing models in *relative* Grothendieck topoi [12] over a presheaf topos that exhibits a phase distinction. In other words, rather than building a site out of predomains in the category of sets, *op. cit.* built an *internal* site based on *internal* predomains in a category of presheaves. Our model of cost-sensitive synthetic domain theory is similar to that of Sterling and Harper [32]. On the other hand, our proof of computational adequacy is different from that of *op. cit.*, as the latter contains a subtle error [30] involving a mismatch between the existential quantifier and the join of a family of Σ-propositions in the lifting of free algebras to formal approximation relations.

Computational adequacy in SDT. Our approach to internal denotational semantics and computational adequacy of $\mathbf{PCF}_{\mathsf{cost}}$ builds on the pioneering work of Simpson [29] on proving the computational adequacy property of **PCF** in elementary topoi models of SDT. One difference between our work and that of *op. cit.* is the addition of the phase distinction, which we use to give an intrinsic denotational account of the interaction between cost and behavior in $\mathbf{PCF}_{\mathsf{cost}}$. Another difference is in the SDT axioms used and the ensuing definition of the dynamic semantics of the object programming language. Simpson [29] assumes a property called *Axiom N* that closes the dominance Σ under countable joins of decidable families in the ambient logic, a fact that we do not rely on in our constructions. The benefit of this axiom is that it enables *op. cit.* to give an internal definition of **PCF** whose dynamic semantics can be characterized by means of existentially quantified statements of the form $\exists[n : \mathbb{N}]\ \phi(n)$ where ϕ is a primitive recursive predicate. This is used to show that a general property of the internal logic of topoi called *1-consistency* [6] is both necessary and sufficient to *externalize* the internal adequacy proof into a corresponding proof in ordinary mathematics. In a follow-up paper, Simpson [28] gave a different logical criterion for the equivalence of internal and external adequacy called *computational 1-consistency* that does not rely on Axiom N. Roughly the idea is to define the programming language and its operational semantics in terms of the *computational natural numbers* (analogous to the predomain $\mathbb{N}_{\mathsf{P}}$ in this paper); computational 1-consistency is just the property needed to ensure that internal computational observations hold externally as well.

By contrast, the dynamic semantics of $\mathbf{PCF}_{\mathsf{cost}}$ in this paper is defined *computationally* and is not known to be equivalent to the operational semantics of Simpson [29] in the absence of Axiom N. However, we expect that a version of our computational semantics using the computational natural numbers will be equivalent to the semantics given in Simpson [28]. On the other hand, we find the computational semantics developed in Section 7 both philosophically and mathematically compelling and deserving of further investigation in its own right. Moreover, although the computational semantics of $\mathbf{PCF}_{\mathsf{cost}}$ does not appear to be definable in terms of countable joins, it can be defined using *synthetic* ω-joins of decidable families. Therefore, we conjecture that one may externalize the internal adequacy proof of $\mathbf{PCF}_{\mathsf{cost}}$ in the manner of Simpson [29] by developing a Kripke-Joyal semantics for the sheaf model defined in Section 9 that unfolds an internal statement involving synthetic ω-joins to an external statement in the metatheory.

11 Conclusion & future work

In this paper we study a language for higher-order recursion $\mathbf{PCF}_{\mathsf{cost}}$ in the setting of synthetic domain theory. Our main contribution is an internal, cost-sensitive version of Plotkin's computational adequacy

[6] A topos $\mathcal{E}$ is *1-consistent* when a closed formula $\exists[n : \mathbb{N}]\ \phi(n)$ of the form described above holding in the internal logic of $\mathcal{E}$ implies that it holds externally.

theorem for $\mathbf{PCF}_{\mathsf{cost}}$. In particular, we define and relate a denotational model of $\mathbf{PCF}_{\mathsf{cost}}$ to a new dynamic semantics for $\mathbf{PCF}_{\mathsf{cost}}$ defined directly in terms of computation that is both natural and mathematically appealing. Here we suggest some ideas for future investigations.

Internal vs. *external adequacy.* In the same vein as the work of Simpson [29,28], we are also interested in giving a logical characterization of when internal computational adequacy (with respect to the computational semantics of $\mathbf{PCF}_{\mathsf{cost}}$) implies external adequacy. However it is not clear to us what would be an analogous condition to 1-consistency: internal notions such as the initial lift algebra ω and synthetic ω-chains do not have natural external counterparts. As mentioned in Section 10, a first step would be to develop a systematic understanding of the logical aspects of the initial lift algebra from an external point of view.

As mentioned in Section 10, one way to obtain external adequacy would be to follow the approach of [28] and define $\mathbf{PCF}_{\mathsf{cost}}$ and its computational semantics purely in terms of computational natural numbers. Alternatively, we may decide to assume Axiom N (see Section 10), which would imply that the computational semantics coincides with ordinary operational semantics in the internal logic of the SDT topos. We do not expect Axiom N to hold in the model we construct in this paper (*cf.* van Oosten and Simpson [34]), but it does not appear to be a limitation of the general approach to the model construction; indeed we believe it should be possible to start with a different domain-theoretic site such that the embedding into the resulting sheaf topos preserves countable coproducts, which would be enough to validate Axiom N.

Cost and information order. As discussed in Section 1.2.3, we would like to combine and develop a practical theory for the interaction of the domain-theoretic information order with a cost preorder in the sense of Grodin *et al.*, who developed a "preorder" version of SDT in which "predomains" are types equipped with a preorder. Following the approach of relative sheaf models of SDT, we conjecture that one may build a model of SDT that further incorporates an intrinsic preorder structure by starting with a domain-theoretic site internal to the category of *simplicial sets*.

Recursive types. We have emphasized recursion at the term level, but synthetic domain theory is also compatible with having recursive types. Simpson [28] has developed in a very general setting the theory and existence of algebraically compact categories of predomains in SDT, and we hope to instantiate the ideas of *op. cit.* at a *relative* sheaf model of SDT similar to the one presented in this paper.

References

[1] Beck, J., *Distributive laws*, in: B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Springer Berlin Heidelberg, Berlin, Heidelberg (1969), ISBN 978-3-540-36091-9.

[2] de Jong, T., *Domain theory in constructive and predicative univalent foundations* (2023). https://doi.org/10.48550/ARXIV.2301.12405

[3] Fiore, M. P., *Lifting as a KZ-doctrine*, in: D. Pitt, D. E. Rydeheard and P. Johnstone, editors, *Category Theory and Computer Science*, pages 146–158, Springer Berlin Heidelberg, Berlin, Heidelberg (1995), ISBN 978-3-540-44661-3.

[4] Fiore, M. P., A. M. Pitts and S. C. Steenkamp, *Quotients, inductive types, and quotient inductive types*, Logical Methods in Computer Science **Volume 18, Issue 2** (2022). https://doi.org/10.46298/lmcs-18(2:15)2022

[5] Fiore, M. P. and G. D. Plotkin, *An extension of models of axiomatic domain theory to models of synthetic domain theory*, in: D. van Dalen and M. Bezem, editors, *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*, volume 1258 of *Lecture Notes in Computer Science*, pages 129–149, Springer (1996). https://doi.org/10.1007/3-540-63172-0_36

[6] Fiore, M. P. and G. Rosolini, *The category of cpos from a synthetic viewpoint*, in: S. D. Brookes and M. W. Mislove, editors, *Thirteenth Annual Conference on Mathematical Foundations of Progamming Semantics, MFPS 1997, Carnegie Mellon University, Pittsburgh, PA, USA, March 23-26, 1997*, volume 6 of *Electronic Notes in Theoretical Computer Science*, pages 133–150, Elsevier (1997). https://doi.org/10.1016/S1571-0661(05)80165-3

[7] Fiore, M. P. and G. Rosolini, *Two models of synthetic domain theory*, Journal of Pure and Applied Algebra **116**, pages 151–162 (1997), ISSN 0022-4049. https://doi.org/10.1016/S0022-4049(96)00164-8

[8] Grodin, H., Y. Niu, J. Sterling and R. Harper, *Decalf: A directed, effectful cost-aware logical framework*, Proceedings of the ACM on Programming Languages **8** (2024).
https://doi.org/10.1145/3632852

[9] Harper, R., J. C. Mitchell and E. Moggi, *Higher-order modules and the phase distinction*, in: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 341–354, Association for Computing Machinery, San Francisco, California, USA (1990), ISBN 0-89791-343-4.
https://doi.org/10.1145/96709.96744

[10] Hoffmann, J., *Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis*, Ph.D. thesis, Ludwig-Maximilians-Universität München (2011).
https://www.cs.cmu.edu/~janh/assets/pdf/Hoffmann11.pdf

[11] Hyland, J. M. E., *First steps in synthetic domain theory*, in: A. Carboni, M. C. Pedicchio and G. Rosolini, editors, *Category Theory*, pages 131–156, Springer Berlin Heidelberg, Berlin, Heidelberg (1991), ISBN 978-3-540-46435-8.

[12] Johnstone, P. T., *Sketches of an Elephant: A Topos Theory Compendium: Volumes 1 and 2*, number 43 in Oxford Logical Guides, Oxford Science Publications (2002).

[13] Kavvos, G. A., E. Morehouse, D. R. Licata and N. Danner, *Recurrence extraction for functional programs through call-by-push-value*, Proceedings of the ACM on Programming Languages **4** (2019).
https://doi.org/10.1145/3371083

[14] Levy, P. B., *Call-by-Push-Value: A Functional/Imperative Synthesis*, Kluwer, Semantic Structures in Computation, 2 (2003), ISBN 1-4020-1730-8.

[15] LONGLEY, J. R. and A. K. SIMPSON, *A uniform approach to domain theory in realizability models*, Mathematical Structures in Computer Science **7**, page 469–505 (1997).
https://doi.org/10.1017/S0960129597002387

[16] Matache, C., S. Moss and S. Staton, *Recursion and Sequentiality in Categories of Sheaves*, in: N. Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:22, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021), ISBN 978-3-95977-191-7, ISSN 1868-8969.
https://doi.org/10.4230/LIPIcs.FSCD.2021.25

[17] Niu, Y. and R. Harper, *A metalanguage for cost-aware denotational semantics*, in: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14 (2023).
https://doi.org/10.1109/LICS56636.2023.10175777

[18] Niu, Y., J. Sterling, H. Grodin and R. Harper, *A cost-aware logical framework*, Proceedings of the ACM on Programming Languages **6** (2022). 2107.04663.
https://doi.org/10.1145/3498670

[19] Paviotti, M., R. E. Møgelberg and L. Birkedal, *A model of PCF in Guarded Type Theory*, Electronic Notes in Theoretical Computer Science **319**, pages 333–349 (2015), ISSN 1571-0661. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).
https://doi.org/10.1016/j.entcs.2015.12.020

[20] Phoa, W., *Domain Theory in Realizability Toposes*, Ph.D. thesis, University of Edinburgh (1991).

[21] Plotkin, G., *A structural approach to operational semantics*, J. Log. Algebr. Program. **60-61**, pages 17–139 (2004).
https://doi.org/10.1016/j.jlap.2004.05.001

[22] Plotkin, G. D., *LCF considered as a programming language*, Theoretical Computer Science **5**, pages 223–255 (1977), ISSN 0304-3975.
https://doi.org/10.1016/0304-3975(77)90044-5

[23] Plotkin, G. D. and J. Power, *Notions of computation determine monads*, in: *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 342–356, Springer-Verlag, Berlin, Heidelberg (2002), ISBN 3-540-43366-X.

[24] Reus, B., *Program Verification in Synthetic Domain Theory*, Ph.D. thesis, Ludwig-Maximilians-Universität München, München (1995).

[25] Reus, B. and T. Streicher, *General synthetic domain theory — a logical approach*, Mathematical Structures in Computer Science **9**, pages 177–223 (1999).
https://doi.org/10.1017/S096012959900273X

[26] Rijke, E., M. Shulman and B. Spitters, *Modalities in homotopy type theory*, Logical Methods in Computer Science **16** (2020).
https://doi.org/10.23638/LMCS-16(1:2)2020

[27] Rosolini, G., *Continuity and effectiveness in topoi*, Ph.D. thesis, University of Oxford (1986).

[28] Simpson, A., *Computational adequacy for recursive types in models of intuitionistic set theory*, Annals of Pure and Applied Logic **130**, pages 207–275 (2004), ISSN 0168-0072. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS).
https://doi.org/10.1016/j.apal.2003.12.005

[29] Simpson, A. K., *Computational Adequacy in an Elementary Topos*, in: G. Gottlob, E. Grandjean and K. Seyr, editors, *Computer Science Logic*, Lecture Notes in Computer Science, pages 323–342, Springer, Berlin, Heidelberg (1999), ISBN 978-3-540-48855-2.
https://doi.org/10.1007/10703163_22

[30] Sterling, J., *Erratum: adequacy of* Sheaf semantics of noninterference (2023).
http://www.jonmsterling.com/jms-005Z.xml

[31] Sterling, J. and R. Harper, *Logical relations as types: Proof-relevant parametricity for program modules*, Journal of the ACM **68** (2021), ISSN 0004-5411. 2010.08599.
https://doi.org/10.1145/3474834

[32] Sterling, J. and R. Harper, *Sheaf semantics of termination-insensitive noninterference*, in: A. P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2022), ISBN 978-3-95977-233-4, ISSN 1868-8969. 2204.09421.
https://doi.org/10.4230/LIPIcs.FSCD.2022.5

[33] Taylor, P., *The fixed point property in synthetic domain theory*, in: *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 152–160 (1991).
https://doi.org/10.1109/LICS.1991.151640

[34] van Oosten, J. and A. K. Simpson, *Axioms and (counter)examples in synthetic domain theory*, Annals of Pure and Applied Logic **104**, pages 233–278 (2000), ISSN 0168-0072.
https://doi.org/10.1016/S0168-0072(00)00014-2

A Properties of the dominance

Proposition A.1 *Assuming Phoa's principle, the dominance Σ is linked.*

Proof. We need to show that the intrinsic order on Σ coincides with the path relation. First, we observe that $\bot \sqsubseteq^\circ \top$; indeed, fixing a map $f : \Sigma \to \Sigma$, by Phoa's principle, evaluation at boundary obtains a pair $(f\ \bot, f\ \top)$ such that $f\ \bot$ implies $f\ \top$. In one direction, suppose $x \sqsubseteq^{\mathsf{p}} y$, which means we have a path $l : \Sigma \to \Sigma$ whose boundary is determined by x, y. Fixing $f : \Sigma \to \Sigma$, we want to show that $f\ x \to f\ y$. But this follows by evaluating $\bot \sqsubseteq^\circ \top$ at the Σ-predicate $f \circ l : \Sigma \to \Sigma$. Conversely, if $x \sqsubseteq^\circ y$, then we have in particular $x \to y$, which by Phoa's principle uniquely determines a path $l : \Sigma \to \Sigma$. □

B Properties of predomains

Proposition B.1 *Any predomain A enjoys the following properties:*

(i) Completeness*: A is orthogonal to $\omega \hookrightarrow \overline{\omega}$.*

(ii) Anti-symmetry*: the intrinsic preorder on A is a partial order.*

(iii) Boundary separation*: maps $\Sigma \to A$ with equal boundary are equal.*

(iv) Linkedness*: the intrinsic preorder and the link relation on A coincide.*

Proof. Replete types are both complete and boundary separated because the latter can be defined by orthogonality conditions and is satisfied by Σ: it is complete by the axioms of SDT (Definition 3.17) and it is boundary separated by Proposition 3.16. Assuming Phoa's principle, the proof that replete types are linked can be found in Taylor [33, Corollary 2.10] and Reus [24, Corollary 6.1.16]. Lastly, one can show that the objects whose link relation is antisymmetric can be defined as an orthogonality condition, thus we know that the link relation on every replete type is antisymmetric, hence it follows the intrinsic order on replete types are also partial orders since they are linked. □

Proposition B.2 *If A has Σ-equality, then so does $\P \vee A$.*

Proof. We observe that $\P \vee A$ is defined as the pushout of the projections of $A \times \P$ as indicated below:

$$\begin{array}{ccc} A \times \P & \longrightarrow & A \\ \downarrow & & \downarrow{\scriptstyle \eta_{\P \vee -}} \\ \P & \xrightarrow[\star]{} & \P \vee A \end{array}$$

Using the fact that A has Σ-equality, we obtain a map $f : (\P \vee A) \times (\P \vee A) \to \P \vee \Sigma$ such that $f(\eta_{\P \vee -}(x), \eta_{\P \vee -}(y)) = \eta_{\P \vee -}(x = y)$ and $f(\star, -) = f(-, \star) = \star$. The desired characteristic map can then be defined as $\sigma \circ f$, where $\sigma : \P \vee \Sigma \to \Sigma$ is defined as follows:

$$\begin{aligned} &\sigma : \P \vee \Sigma \to \Sigma \\ &\sigma(\eta_{\P \vee -}(\phi)) = \P \vee \phi \\ &\sigma(\star) = \top \end{aligned}$$

For any $u, v : \P \vee A$, if $u = v = \eta_{\P \vee -}(x)$ for some $x : A$, then we have $\sigma(f(u, v)) = \sigma(\eta_{\P \vee -}(x = x)) = \P \vee (x = x) = \top$. Otherwise, we have that u or v is $\star$, which means $\P$ holds and so $\sigma(f(u, v)) = \sigma(\star) = \top$ as well. Conversely, suppose $\sigma(f(u, v)) = \top$, and that $u = \eta_{\P \vee -}(x)$ and $v = \eta_{\P \vee -}(y)$, which means that $\sigma(\eta_{\P \vee -}(x = y)) = \P \vee (x = y)$ holds. If $\P$ holds, we are done as $(\P \vee A) \cong 1$ in this case. Otherwise, we have $x = y$, and so $u = \eta_{\P \vee -}(x) = \eta_{\P \vee -}(y) = v$. Lastly, if either u or v is the unique element $\star$ then we may discharge the case as above. □

Proposition B.3 *When A and B are predomains, the synthetic orders on $A \times B$ and $A \to B$ are pointwise.*

Proof. This is Proposition 5.4.4 of Phoa [20]. We just show the case for the function types. Given a path $f \sqsubseteq_{X \to B} g$, it is clear that we may construct a path $f\ x \sqsubseteq Bg\ x$ for all $x : X$. Conversely, suppose we are given a path $f\ x \sqsubseteq Bg\ x$ for all $x : X$. By Proposition 4.1, B is boundary separated, and so such paths are necessarily unique, and so we have a function $\alpha : X \to \Sigma \to B$ such that $\alpha(x)$ is a path $f\ x \sqsubseteq Bg\ x$. We then obtain a path $f \sqsubseteq_{X \to B} g$ by taking the exponential transpose of α. □

Proposition B.4 *Given a predomain A, we have that $x \sqsubseteq_{\mathsf{L}A} y$ if and only if $x{\downarrow}$ implies $y{\downarrow}$ and whenever $x{\downarrow}$, we have $x \sqsubseteq_A y$.*

Proof. In the forward direction, we have a path $x{\downarrow} \sqsubseteq y{\downarrow}$, which means $x{\downarrow}$ implies $y{\downarrow}$ as the Σ is linked by Proposition 3.13. Suppose $x{\downarrow}$, and let $f : A \to \Sigma$ be arbitrary. By assumption, we have that $f'(x) \to f'(y)$, where $f'((\phi, u)) = \phi \angle f \circ u$. In other words, we have $x{\downarrow} \angle f(x)$ implies $y{\downarrow} \angle f(y)$. Since $x{\downarrow}$ holds, we have that $f(x) \to f(y)$, which by definition means $x \sqsubseteq_A y$.

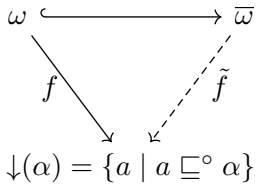
In the backward direction, let $\alpha : x{\downarrow} \to x \sqsubseteq_A y$ be the given partial path. We may define a total path $\beta : \Sigma \to \mathsf{L}A$ between x and y by setting $\beta(\phi) = (x{\downarrow}, \lambda p.\ \alpha\ p\ \phi)$. Thus we have $x \sqsubseteq_{\mathsf{L}A} y$ as required. □

Proposition B.5 *For every map $f : \omega \to A$ into a complete type A, there exists an element $a_\infty : A$ such that a_∞ is a least upper bound of f with respect to the intrinsic preorder.*

Proof. The final L-coalgebra is equipped with a global element $\infty : \overline{\omega}$ that can be thought of as the "point at infinity". Define f_∞ be the element determined by the unique extension $\overline{f} : \overline{\omega} \to A$ evaluated at the invariant point ∞.

(i) First we show that f_∞ is an upper bound for f. Fixing $i : \omega$, we need to show that $f\ i \sqsubseteq^\circ_A f_\infty$. Because $\overline{f}$ extends f, it suffices to show $\overline{f}\ i \sqsubseteq^\circ_A f_\infty$. Using the fact that every map is monotone with respect to the specialization order, the result holds because $i \sqsubseteq^\circ_{\overline{\omega}} \infty$.

(ii) Let α be an upper bound for f. We need to show that $f_\infty \sqsubseteq^\circ \alpha$. If the principal lower set $\downarrow(\alpha)$ is complete, we have the following lifting situation:



In the above $\tilde{f}$ is the unique extension of f considered as a map $\omega \to \downarrow(\alpha)$. By uniqueness of $\overline{f}$ as the extension of $f : \omega \to A$, $\tilde{f}$ is equal to $\overline{f}$ considered as maps $\overline{\omega} \to A$. Consequently we have that $f_\infty = \overline{f}(\infty) = \tilde{f}(\infty)$, so the result follows by observing that $\tilde{f}(\infty) \in \downarrow(\alpha)$.

It remains to show that $\downarrow(\alpha)$ is complete. We can express the principal lower set as follows:

$$\begin{aligned}\downarrow(\alpha) &= \{a \mid a \sqsubseteq^\circ \alpha\} \\ &= \{a \mid \forall f : A \to \Sigma\ f(a) \to f(\alpha)\} \\ &= \bigcap_{f : A \to \Sigma} \{a \mid f(a) \to f(\alpha)\}\end{aligned}$$

Because complete types are internally complete, the result would follow if we can show that $S = \{a \mid f(a) \to f(\alpha)\}$ is complete. We may show that S can be computed as follows:

$$\begin{array}{ccc} S & \longrightarrow & A \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \langle f, f(\alpha)\rangle} \\ \Sigma^\Sigma & \xrightarrow[\partial]{} & \Sigma \times \Sigma \end{array}$$

Since S can be defined as the limit of a diagram of complete types, it is complete as well.

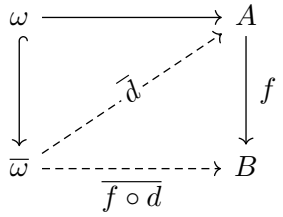
□

Proposition B.6 *Every map $f : A \to B$ between predomains is monotone in the synthetic order.*

Proof. Given $a \sqsubseteq b$, we derive a path $\Sigma \to B$ whose boundary is $(f\ a, f\ b)$ by postcomposing with f, and so $f\ a \sqsubseteq f\ b$ as well. □

Proposition B.7 *Every map $f : A \to B$ between predomains is continuous in the sense that $f(\bigvee d) = \bigvee(f \circ d)$ for every synthetic ω-chain $d : \omega \to A$.*

Proof. Predomains are complete, so we have the following extensions of d and $f \circ d$:



Because extensions along $\omega \hookrightarrow \overline{\omega}$ are unique for complete types, we have $f \circ \overline{d} = \overline{f \circ d}$. But by definition of the synthetic ω-join, this means that $f(\bigvee d) = f(\overline{d}(\infty)) = \overline{f \circ d}(\infty) = \bigvee(f \circ d)$. □

Proposition B.8 *The intersection of a family of admissible subsets of a domain is admissible.*

Proof. The least element is contained in the intersection as it is contained in every fiber. Suppose that $f : \omega \to A$ is a synthetic ω-chain such that $f_i \in \bigcap F$. But since $f_i \in F_j$ for every $j : J$, we have $\bigvee f \in F_j$ as every F_j is admissible, which means $\bigvee f \in \bigcap F$ as well. □

Proposition B.9 *If P, Q are Σ-subsets of a domain A, then the exponential subobject Q^P is an admissible subset of A.*

Proof. Let $f_i \in Q^P$ for some synthetic ω-chain f. Suppose that $\bigvee f \in P$. We need to show that $\bigvee f \in Q$. By the universal property of $\bigvee$, we may assume that $f_i \in P$ for some $i : \omega$. By assumption, this means that $f_i \in Q$ and thence $\bigvee f \in Q$ as every Σ-subset is monotone in the synthetic order. □

C Properties of the denotational semantics

Proposition C.1 *The action of the derived cost algebra satisfies the following equations for $e : \mathsf{T}(A)$ and $f : A \to X$ for some $\mathbb{T}$-algebra X:*

$$c \boxplus_{\mathsf{T}(A)} e = (e{\downarrow}, \lambda u.\ \mu_{\mathbb{C}\times -}(c, e))$$
$$f^\sharp(c \boxplus_{\mathsf{T}(A)} e) = c \boxplus_X (f^\sharp\ e)$$
$$(c \boxplus_{A\to X} f)\ a = c \boxplus_X (f\ a)$$

Proof. Routine computation using the distributive law τ of $\mathbb{C} \times -$ over the lifting monad. □

D Properties of the computational semantics

Proposition D.1 *The relation $\pi_1 \circ \mathsf{eval}$ is functional,* i.e. *$\downarrow\mathsf{eval}(e, v)$ and $\downarrow\mathsf{eval}(e, v')$ implies $v = v'$.*

Proof. Consider the following subset:

$$P = \left\{\alpha \mid \forall[e, v, v']\ \alpha(e, v){\downarrow} \wedge \alpha(e, v'){\downarrow} \to v = v'\right\}$$

We may check that P is admissible and proceed by fixed-point induction. Suppose that $\alpha \in P$ and that $\Phi_{\mathsf{eval}}(\alpha)(e, v){\downarrow}$ and $\Phi_{\mathsf{eval}}(\alpha)(e, v'){\downarrow}$. We need to show that $v = v'$. We proceed by cases on $\mathsf{out}(e)$.

(i) If $e \mapsto c', e'$, we may deduce that $\alpha(e', v)\downarrow$ and $\alpha(e', v')\downarrow$, so the result follows from the assumption that $\alpha \in P$.

(ii) Otherwise, we have that $e = v$ and $e = v'$ by definition of Φ_{eval}, and so $v = v'$.

□

Proposition D.2 *If* $\mathsf{eval}(e, \mathsf{ret}(v)) = c_1$ *and* $\mathsf{eval}(g\ v, \mathsf{ret}(w)) = c_2$, *then* $\mathsf{eval}(e; g, \mathsf{ret}(w)) = c_1 + c_2$.

Proof. Consider the following subset of $\Pi_{A:\mathsf{tp}}.\ \mathsf{UF}A \to \mathsf{UF}A \to \mathsf{T}(1)$:

$$P = \{\alpha \mid \forall[e]\ \alpha(e, v)\downarrow \wedge \mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow \to \mathsf{eval}(e; g, \mathsf{ret}(w)) = \alpha(e, v) + \mathsf{eval}(g\ v, \mathsf{ret}(w))\}$$

It suffices to show that $\mathsf{eval} \in P$. Observing that P is admissible, we proceed by fixed-point induction. Suppose that $\alpha \in P$, $\Phi_{\mathsf{eval}}(\alpha)(e, v)\downarrow$, and that $\mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow$. We need to show that $\mathsf{eval}(e; g, \mathsf{ret}(w)) = \Phi_{\mathsf{eval}}(\alpha)(e, v) + \mathsf{eval}(g\ v, \mathsf{ret}(w))$. We proceed by cases on $\mathsf{out}(e)$.

(i) If $e \mapsto c', e'$, then we compute:

$$\begin{aligned}\mathsf{eval}(e; g, \mathsf{ret}(w)) &= c' \boxplus \mathsf{eval}(e'; g, \mathsf{ret}(w)) \\ &= c' + \alpha(e', v) + \mathsf{eval}(g\ v, \mathsf{ret}(w)) \\ &= \Phi_{\mathsf{eval}}(\alpha)(e, v) + \mathsf{eval}(g\ v, \mathsf{ret}(w))\end{aligned}$$

Where the first equality follows from the assumption that $\alpha \in P$ and the second by the definition of Φ_{eval}.

(ii) Otherwise, we have that $e\ \mathsf{val}$. Since $\Phi_{\mathsf{eval}}(\alpha)(e, \mathsf{ret}(v))\downarrow = (e = \mathsf{ret}(v), 0)\downarrow = (e = \mathsf{ret}(v))$ holds, we can compute:

$$\begin{aligned}\mathsf{eval}(e; g, \mathsf{ret}(w)) &= \mathsf{eval}(\mathsf{ret}(v); g, \mathsf{ret}(w)) \\ &= \Phi_{\mathsf{eval}}(\mathsf{eval})(\mathsf{ret}(v); g, \mathsf{ret}(w)) \\ &= \mathsf{eval}(g\ v, \mathsf{ret}(w))\end{aligned}$$

But this is what we needed to show since $\Phi_{\mathsf{eval}}(\alpha)(e, \mathsf{ret}(v)) = 0$.

□

Proposition D.3 *The following inference rule is valid for any* Σ*-predicate* φ*:*

$$\frac{\forall[v : A]\ \mathsf{eval}(e, v)\downarrow \wedge \mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(e, v) + \mathsf{eval}(g\ v, \mathsf{ret}(w)))}{\mathsf{eval}(e; g, \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(e; g, \mathsf{ret}(w)))}$$

Proof. Consider the subset $P \subseteq \mathsf{UF}A \to \mathsf{UF}A \to \mathsf{T}(1)$ defined as the intersection of the following subsets:

$$\begin{aligned}Q &= \{\alpha \mid \alpha \sqsubseteq \mathsf{eval}\} \\ R &= \left\{\alpha \mid \forall\left[c', e', n\right]\ (e \mapsto^n c', e') \wedge \left(\alpha(e'; g, \mathsf{ret}(w))\downarrow\right) \to \varphi(c' + \alpha(e'; g, \mathsf{ret}(w)))\right\}\end{aligned}$$

It suffices to show that $\mathsf{eval} \in P$. We have that P is admissible, and we proceed by fixed-point induction. Suppose that $\alpha \in P$. We need to show that $\Phi_{\mathsf{eval}}(\alpha) \in P$, where Φ_{eval} is the characteristic functional of eval (Section 7.1). It's immediate that $\Phi_{\mathsf{eval}}(\alpha) \in Q$. It remains to show that it is also contained in R. So suppose that $e \mapsto^n c', e'$ and $\Phi_{\mathsf{eval}}(\alpha)(e'; g, \mathsf{ret}(w))\downarrow$. We want to show that $\varphi(c' + \Phi_{\mathsf{eval}}(\alpha)(e'; g, \mathsf{ret}(w)))$. We proceed by cases on $\mathsf{out}(e')$.

(i) If $\mathsf{out}(e') = \mathsf{inl}{\cdot}\star$, then we know that $e' = \mathsf{ret}(v)$ for some $v : \mathsf{UF1}$. Stepping the operational semantics, we have that $\mathsf{ret}(v); g \mapsto 0, g\ v$, and by definition of the computational semantics $\Phi_{\mathsf{eval}}(\alpha)(e'; g, \mathsf{ret}(w)) = 0 \boxplus \alpha(g\ v, \mathsf{ret}(w)) = \alpha(g\ v, \mathsf{ret}(w))$. Since we assumed $\alpha \in Q \iff \alpha \sqsubseteq \mathsf{eval}$, we also have $\mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow$, and since $\mathbb{C}$ is discrete we have $\alpha(g\ v, \mathsf{ret}(w)) = \mathsf{eval}(g\ v, \mathsf{ret}(w))$. Recalling the premise and the fact that $\mathsf{eval}(e, \mathsf{ret}(v)) = c'$, we may conclude that $\varphi(c' + \mathsf{eval}(g\ v, \mathsf{ret}(w)))$, which is what we needed to show.

(ii) Otherwise, $\mathsf{out}(e') = \mathsf{inr} \cdot (c'', e'')$ for some $e'' : \mathsf{UF1}$, and we have that $e'; g \mapsto c'', e''; g$. By definition of the computational semantics, this means that $\Phi_{\mathsf{eval}}(\alpha)(e'; g, \mathsf{ret}(w)) = c'' \boxplus \alpha(e''; g, \mathsf{ret}(w))$. Since we assumed that $c'' \boxplus \alpha(e''; g, \mathsf{ret}(w))\downarrow$, we can use the laws of the derived algebra (Proposition 6.1) to deduce that $\alpha(e''; g, \mathsf{ret}(w))\downarrow$ as well, and so by the assumption that $\alpha \in R \subseteq P$, we have that $\varphi(c' + c'' + \alpha(e''; g, \mathsf{ret}(w)))$ holds, which is what we needed to show.

□

Proposition D.4 *We have* $\mathsf{profile}((e; g); i) = \mathsf{profile}(e; (\lambda v.\ g\ v; i))$.

Proof. In one direction, we show that $\mathsf{profile}((e; g); i)\downarrow$ implies $\mathsf{profile}(e; (\lambda v.\ g\ v; i))\downarrow$ and both denote identical costs. Consider the Σ-predicate φ such that $\varphi(c)$ if and only if $\mathsf{profile}(e; (\lambda v.\ g\ v; h)) = c$. Suppose that $\mathsf{eval}(e; g, \mathsf{ret}(w))\downarrow$ and $\mathsf{profile}(i\ w)\downarrow$. By computational induction on sequencing Proposition 7.4, it suffices to show that $\mathsf{profile}(e; (\lambda v.\ g\ v; h)) = \mathsf{eval}(e; g, \mathsf{ret}(w)) + \mathsf{profile}(i\ w)$. Applying computational induction on $\mathsf{eval}(e; g, \mathsf{ret}(w))\downarrow$, we further suppose that $\mathsf{eval}(e, \mathsf{ret}(v))\downarrow$ and $\mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow$ and aim to show that $\mathsf{profile}(e; (\lambda v.\ g\ v; h)) = \mathsf{eval}(e, \mathsf{ret}(v)) + \mathsf{eval}(g\ v, \mathsf{ret}(w)) + \mathsf{profile}(i\ w)$.

(i) We claim that $\mathsf{profile}(e; (\lambda v.\ g\ v; i))\downarrow$. By the big-step semantics of profiling Corollary 7.3, it suffices to show that $\mathsf{eval}(e, \mathsf{ret}(v))\downarrow$ for some v and $\mathsf{profile}(g\ v; i)\downarrow$. The former follows from our assumption; for the latter, it suffices to show that $\mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow$ and $\mathsf{profile}(i\ w)\downarrow$, both of which follow from assumptions.

(ii) Given that $\mathsf{profile}(e; (\lambda v.\ g\ v; i))\downarrow$, we may apply computational induction again: supposing that $\mathsf{eval}(e, \mathsf{ret}(v'))\downarrow$ and $\mathsf{profile}(g\ v; i)\downarrow$, we have to show that $\mathsf{eval}(e, \mathsf{ret}(v')) + \mathsf{profile}(g\ v; i) = \mathsf{eval}(e, \mathsf{ret}(v)) + \mathsf{eval}(g\ v, \mathsf{ret}(w)) + \mathsf{profile}(i\ w)$, which follows from the uniqueness of evaluation Proposition 7.1 and big-step semantics of profiling Corollary 7.3.

In the other direction, suppose that $\mathsf{profile}(e; (\lambda v.\ g\ v; i))\downarrow$. It suffices to show that $\mathsf{profile}((e; g); i)\downarrow$. By computational induction, we may assume that $\mathsf{eval}(e, \mathsf{ret}(v))\downarrow$ and $\mathsf{profile}(g\ v; i)\downarrow$. Applying computational induction again, we can also assume that $\mathsf{eval}(g\ v, \mathsf{ret}(w))\downarrow$ and $\mathsf{profile}(i\ w)\downarrow$ for some w. By the big-step semantics of profiling Corollary 7.3, it suffices to show that $\mathsf{eval}(e; g, \mathsf{ret}(w))\downarrow$ and $\mathsf{profile}(i\ w)\downarrow$. The latter is our assumption, and the former follows from the big-step semantics of evaluation Proposition 7.2. □

Proposition D.5 *The following is valid:*

$$\frac{\forall[e]\ \mathsf{eval}(f, \lambda e)\downarrow \wedge \mathsf{eval}(e[v], \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(f, \lambda e) + \mathsf{eval}(e[v], \mathsf{ret}(w)))}{\mathsf{eval}(f\ v, \mathsf{ret}(w))\downarrow \to \varphi(\mathsf{eval}(f\ v, \mathsf{ret}(w)))}$$

Proposition D.6 *We have that* $\mathsf{eval}((e; g)\ w, z) = \mathsf{eval}(e; \lambda v.\ g\ v\ w, z)$.

E Soundness of the denotational semantics

Proposition E.1 *If* $e \mapsto c, e'$, *then* $[\![e]\!] = c \boxplus [\![e']\!]$.

Proof. By induction on the derivation of $e \mapsto c, e'$. □

Theorem E.2 *If* $\mathsf{eval}(e, v)\downarrow$, *then* $[\![e]\!] = \mathsf{eval}(e, v) \boxplus [\![v]\!]$.

Proof. Consider the following subset:

$$P = \{\alpha \mid \forall[e]\ \alpha(e, v)\downarrow \to [\![e]\!] = \mathsf{eval}(e, v) \boxplus [\![v]\!]\}$$

Because $[\![e]\!] = \mathsf{eval}(e, v) \boxplus [\![v]\!]$ is a Σ-proposition, we see that P is an admissible subset. Suppose that $\alpha \in P$ and $\Phi_{\mathsf{eval}}(\alpha)(e, v)\downarrow$. We need to show that $[\![e]\!] = \mathsf{eval}(e, v) \boxplus [\![v]\!]$. We proceed by cases on $\mathsf{out}(e)$.

(i) If $e \mapsto c', e'$, then by the soundness of the one step relation Proposition 8.1, it suffices to show that $c' \boxplus [\![e']\!] = c' \boxplus \mathsf{eval}(e', v) \boxplus [\![v]\!]$, which follows from the assumption, noting that $\Phi_{\mathsf{eval}}(\alpha)(e, v)\downarrow$ implies $\alpha(e, v)\downarrow$.

(ii) Otherwise, we have that $e = v$, and so the result holds since $\mathsf{eval}(e, e) = 0$.

□

F Proofs for the computational adequacy property

Proposition F.1 *If $d \lhd_X e$ and $e' \mapsto c, e$, then $c \boxplus d \lhd_X e'$.*

Proof. By induction on X, using the laws of the cost algebraProposition 6.1. □

F.1 Admissibility

Proposition F.2 *We have that $- \lhd_{\mathsf{U}\mathsf{F}A} e$ is an admissible subset of $\mathsf{T}(A)$.*

Proof. First, we have to show that $\bot \lhd_{\mathsf{U}\mathsf{F}A} e$. By definition, this means to show $f^\sharp(\bot) \sqsubseteq \mathsf{profile}(e; g)$ for all f $(A \Rightarrow \mathsf{adq})$ g. But this holds since $f^\sharp(\bot) = \bot$ and $\bot$ is the least element of $\mathsf{T}(A)$. Otherwise, let d be a synthetic ω-chain such that $d_i \lhd_{\mathsf{U}\mathsf{F}A} e$ for all $i : \omega$. We want to show that $\bigvee d \lhd_{\mathsf{U}\mathsf{F}A} e$, which is to show that $f^\sharp(\bigvee d) \sqsubseteq \mathsf{profile}(e; g)$ for all f $(A \Rightarrow \mathsf{adq})$ g. Since $f^\sharp(\bigvee d) = \bigvee(f^\sharp \circ d)$, this means to show $\bigvee(f^\sharp \circ d) \sqsubseteq \mathsf{profile}(e; g)$. By the universal property of the synthetic ω-supremum, it suffices to show $f^\sharp(d_i) \sqsubseteq \mathsf{profile}(e; g)$ for all $i : \omega$, but this is the assumption. □

Proposition F.3 *Suprema of synthetic ω-chains in function spaces are computed pointwise.*

Proposition F.4 *Given that $- \lhd_{\mathsf{U}X} e$ is admissible for all $e : \mathsf{U}X$, we have that $- \lhd_{\mathsf{U}(A \to X)} e$ is as well.*

Proof. Because $\bot(a) = \bot$, we have that $\bot \lhd_{\mathsf{U}(A \to X)} e$. Suppose that $f_i \lhd_{\mathsf{U}(A \to X)} e$. We need to show that $\bigvee f \lhd_{\mathsf{U}(A \to X)} e$. Suppose that $a \lhd_A b$. We need to show that $(\bigvee f)\ a \lhd_{\mathsf{U}X} e\ b$. This follows from the characterization of synthetic ω-suprema in function spaces (Proposition F.3) and the assumption that $- \lhd_{\mathsf{U}X} e$. □

Proposition F.5 *Given $e : \mathsf{U}X$, we have that $- \lhd_{\mathsf{U}X} e$ is an admissible subset of $[\![\mathsf{U}X]\!]$.*

Proof. By Propositions F.2 and F.4. □

F.2 Fundamental lemma

We give the representative cases of the proof by induction on the derivation of terms.

Lemma F.6 *If $a \lhd_A v$, then $\eta_{\mathsf{T}}(a) \lhd_{\mathsf{U}\mathsf{F}A} \mathsf{ret}(v)$.*

Proof. Let f $(\lhd_A \Rightarrow \mathsf{adq})$ g. We need to show that $(f^\sharp(\eta_{\mathsf{T}}(a)))$ adq $(\mathsf{ret}(v); g)$. Computing the denotational semantics and applying Proposition F.1, it suffices to show that $(f\ a)$ adq $(g\ v)$, which follows from our assumption. □

Lemma F.7 *If $d \lhd_{\mathsf{U}\mathsf{F}A} e$ and $f \lhd_{\mathsf{U}(A \to X)} g$, then $f^\sharp(d) \lhd_{\mathsf{U}X} e; g$.*

Proof.

By induction on X.

(i) If $X = \mathsf{F}B$, let h $(\lhd_B \Rightarrow \mathsf{adq})$ i. We need to show that $h^\sharp(f^\sharp(d))$ adq $(e; g); i$. Computing the denotational semantics and using the fact that we may reassociate sequences (Proposition 7.5), it suffices to show $((h^\sharp \circ f)^\sharp(d))$ adq $(e; (\lambda v.\ g\ v; i))$. By the assumption that $d \lhd_{\mathsf{U}\mathsf{F}A} e$, it suffices to show that for all $a \lhd_A v$, we have that $(h^\sharp(f\ a))$ adq $(g\ v; i)$, which follows directly from the assumptions that $f \lhd_{\mathsf{U}(A \to X)} g$ and h $(\lhd_B \Rightarrow \mathsf{adq})$ i.

(ii) If $X = B \to Y$, suppose that $b \lhd_B v$. We need to show that $(f^\sharp(d))\ b \lhd_{\mathsf{U}Y} (e; g)\ v$. Unraveling the denotational semantics and the computational semantics (using Proposition D.6), it suffices to show $(\lambda d.\ f\ d\ b)^\sharp\ d \lhd_{\mathsf{U}Y} (e; \lambda d.\ g\ d\ v)$, which follows from the inductive hypothesis and the assumption that $f \lhd_{\mathsf{U}(A \to (B \to Y))} g$.

□

Lemma F.8 *If $d \lhd_X e$, then $c \boxplus d \lhd_X \mathsf{step}^c(e)$.*

Proof. Since $\mathsf{step}^c(e) \mapsto c, e$, the result holds by Proposition F.1. □

Theorem F.9 *For every closed term $e : \Gamma \vdash A$, the approximation $\Gamma \vdash [\![e]\!] \lhd_A e$ holds.*

Proof. By Lemmas F.6 to F.8 and Proposition F.5. □

Theorem F.10 *Any $e : \mathsf{UF1} \rightharpoonup \mathsf{F2}$ is weakly, extensionally constant in the sense that for all $x, y : \mathsf{UF1}$, if $\mathsf{profile}(x)\downarrow$ and $\mathsf{profile}(y)\downarrow$, then $\mathsf{eval}(e\ x, \mathsf{ret}(v))\downarrow$ and $\mathsf{eval}(e\ y, \mathsf{ret}(u))\downarrow$ imply $v = u$.*

Proof. Let $c : \mathbb{C}$ and $d : \mathbb{C}$ be the costs denoted by $\mathsf{eval}(e\ x, \mathsf{ret}(v))$ and $\mathsf{eval}(e\ y, \mathsf{ret}(u))$. By soundness Theorem 8.2 and laws of the derived algebra Proposition 6.1, we have that $[\![e\ x]\!] = c \boxplus \eta_{\mathbb{T}}([\![v]\!]) = \eta_{\mathbb{L}}(c, [\![v]\!])$ and similarly $[\![e\ y]\!] = \boxplus\eta_{\mathbb{T}}([\![u]\!]) = \eta_{\mathbb{L}}(d, [\![u]\!])$. It suffices to show that $[\![v]\!] =_2 [\![u]\!]$. Because 2 is a purely extensional type (as required by Definition 3.17), we may assume that ¶ holds. By assumption and soundness, we have $[\![x]\!] = [\![y]\!]$, and so $[\![e\ x]\!] = \eta_{\mathbb{L}}(c, [\![v]\!]) = \eta_{\mathbb{L}}(d, [\![u]\!]) = [\![e\ y]\!]$, which means that $[\![v]\!] = [\![u]\!]$ since $c = d$ as elements of a purely intensional type $\mathbb{C}$. □