Sequential Functions on Indexed Domains and Full Abstraction for a Sub-language of PCF

Stephen Brookes Shai Geva

April 1993
CMU-CS-93-163

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in Proceedings of *Mathematical Foundations of Programming Semantics*, New Orleans, 1993 (Springer Verlag Lecture Notes in Computer Science).

Abstract

We present a general semantic framework of sequential functions on domains equipped with a parameterized notion of incremental sequential computation. Under the simplifying assumption that computation over function spaces proceeds by successive application to constants, we construct a sequential semantic model for a non-trivial sub-language of PCF with a corresponding syntactic restriction — that variables of function type may only be applied to closed terms. We show that the model is fully abstract for the sub-language, with respect to the usual notion of program behavior.

This research was supported in part by National Science Foundation grant CCR-9006064.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. government.

Keywords: theory, applicative (functional) programming, semantics, topology, sequentiality

1 Introduction

A semantics for a programming language is fully abstract with respect to a given notion of program behavior iff the semantics distinguishes between two terms exactly when there is a program context in which the terms induce different behavior. Intuitively, a fully abstract semantics is at precisely the right level of abstraction to support compositional reasoning about behavior. It has turned out to be surprisingly difficult to give natural (*i.e.*, language-independent) constructions of fully abstract semantic models for sequential languages such as PCF [Plo77, BCL85]. The known constructions of fully abstract models for PCF [Mil77, Ber78, Mul87] are not natural, yet there are natural fully abstract models for an extension of PCF with parallel facilities [Plo77] and, more recently, with control facilities [CF92, Cur92]. There is currently no definition of sequential functions suitable for defining a natural extensional semantic model for PCF.

The first definitions of sequential functions, given by Milner [Mil77] and Vuillemin [Vui73], were limited to functions on products of flat domains. Kahn and Plotkin [KP78] introduced concrete data structures and concrete domains, and defined sequential functions between concrete domains. However, the sequential functions between two concrete domains do not form a concrete domain (under either the pointwise or stable orders). Berry introduced dI-domains, stable functions and the stable ordering [Ber78]; the stable functions between two dI-domains, ordered stably, form a dI-domain. However, the stable functions do not provide the desired notion of sequential functions, since some stable functions are not sequential. Berry and Curien [BC82, Cur86] defined sequential algorithms between concrete domains, and obtained a sequential intensional model from which one may recover the Kahn-Plotkin sequential functions by taking an extensional quotient. More recently, Bucciarelli and Ehrhard [BE91] introduced a notion of strongly stable functions between qualitative domains equipped with a coherence structure (QDC's), generalizing the Kahn-Plotkin definition. In earlier work [BG92] we defined sequential functions on Scott domains that generalized Kahn and Plotkin's sequential functions, and we obtained several closure results under the sequential function space.

We continue here the investigation of sequentiality. We present a framework of *indexed domains*, domains equipped with a parameterized notion of incremental sequential computation, formulated as an *index structure*. We give a general definition of *sequential functions between indexed domains*, as continuous functions that, essentially, respect the index structure. We define an indexed domain product, and show closure of indexed domains under the sequential function space, for both the pointwise and the stable orderings (with different classes of underlying domains). Indexed domains are closely related to Bucciarelli and Ehrhard's domains with coherence structures; we discuss this relationship in the conclusion.

Our earlier definition of sequential functions [BG92] arises when all domains are equipped with a particular *data-index structure*, which imposes a notion of incremental computation adequate for domains of data. This is the index structure which is (implicitly) present in Kahn and Plotkin's, Milner's and Vuillemin's definitions of sequentiality, as well as in Berry and Curien's sequential algorithms over concrete data structures and the language CDS0 [BC85, Cur86]; it is also used by Bucciarelli and Ehrhard for defining sequentiality at first-order. In PCF, however, computation over function spaces proceeds in an inherently different manner, and thus the use of data-indices is not always appropriate; a suitable higher-order notion of incremental computation is called for.

The framework of indexed domains and sequential functions is not adequate to provide a fully abstract model for PCF, since function application fails to be sequential. Nevertheless, by analyzing higher-order computation, we are able to arrive at a key simplifying assumption. Application of a sequential function to fixed arguments is in fact a sequential function. We may therefore restrict our

attention to higher-order computations that proceed by successive applications to constants, and we obtain a new higher-order notion of *constant-applicative sequentiality*. Importantly, the semantic restriction can be mirrored by a syntactic restriction on PCF — we restrict the application of variables of functional type so that they are only applied to arguments that are independent of the input, *i.e.*, to closed terms. We thus obtain ca-PCF, a non-trivial sub-language of PCF. Finally, the operational content of the indices allows us to tie the knot and show that the sequential model we have built, employing data sequentiality at ground types and the pointwise order and constant-applicative sequentiality at arrow types, is fully abstract for ca-PCF, with respect to the usual notion of program behavior.

2 Preliminaries

We assume conventional domain-theoretic definitions and notations. The original definitions of stability are due to Berry [Ber78], and Zhang [Zha91] gave a generalized topological characterization. We generalized Zhang's definitions to Scott domains and to the pointwise order in [BG92], where a full development may be found, as well as a fuller treatment of FM-domains.

A *Scott domain* is a directed-complete, bounded-complete, ω-algebraic poset with a least element. We write $x \Uparrow y$ to indicate that x and y are bounded (consistent). We write D_{fin} for the set of isolated elements of D. A *dI-domain* is a distributive Scott domain with property (I), *i.e.*, such that every isolated element dominates finitely many elements. A subset X of a poset is *down-directed* iff every pair of elements of X has a lower bound in X. The *covering* relation is defined by setting $x \prec\!\!\!\cdot\; y$ iff $x < y$ and the set $\{z \mid x < z \,\&\, z < y\}$ is empty. We define the up-closure of $x \in D$ by $\mathsf{up}\, x = \{z \in D \mid x \leq z\}$. For $u \subseteq D$ let $\mathsf{up}\, u = \bigcup \{\mathsf{up}\, x \mid x \in u\}$. A set u is up-closed iff $u = \mathsf{up}\, u$. A down-closure operation down is defined dually.

An *FM-domain* is a Scott domain with the finite meet property (FM): the meet of each pair (or equivalently, every non-empty finite set) of isolated elements is itself isolated[1]. FM-domains are a proper intermediate class of domains between dI-domains and Scott domains.

In an algebraic poset, a subset $p \subseteq D$ is *Scott open* iff $p = \mathsf{up}(p \cap D_{\mathrm{fin}})$, and it is *stable open* if, in addition, it is closed under bounded meets, *i.e.*, if $x_1, x_2 \in p$ and $x_1 \Uparrow x_2$ then $x_1 \wedge x_2 \in p$. Write $\mathsf{Sc}\, D$ for the set of Scott opens of D and $\mathsf{St}\, D$ for the set of stable opens of D. For every $x \in D_{\mathrm{fin}}$, $\mathsf{up}\, x$ is Scott open and stable open. The Scott opens and the stable opens of a domain D have T_0 separation, *i.e.*, for every $x, y \in D$, $x = y$ iff $\{p \in \mathsf{Sc}\, D \mid x \in p\} = \{p \in \mathsf{Sc}\, D \mid y \in p\}$, and likewise for stable opens.

Scott opens define the Scott topology. Stable opens do not form a true topology, but may be regarded as a generalized topology. Every stable open may be decomposed into a disjoint union of *lobes*, which are downwards-directed Scott opens. In a dI-domain every lobe has a least element. Stable opens of a dI-domain are therefore up-closures of pairwise inconsistent sets of isolated elements, coinciding with Zhang's stable neighborhoods [Zha91].

A function $f : D \to D'$ is Scott continuous, or just *continuous*, iff $f^{-1}q \in \mathsf{Sc}\, D$ for every $q \in \mathsf{Sc}\, D'$. Equivalently, f is continuous iff it is monotone and preserves directed lubs. A function $f : D \to D'$ is stable continuous, or just *stable*, iff $f^{-1}q \in \mathsf{St}\, D$ for every $q \in \mathsf{St}\, D'$. Equivalently, f is stable iff f is continuous and preserves bounded meets, *i.e.*, if $x_1 \Uparrow x_2$ then $f(x_1 \wedge x_2) = f(x_1) \wedge f(x_2)$.

For continuous functions $f, g : D \to D'$, we define the pointwise ordering by $f \leq g$ iff $fx \leq gx$ for every $x \in D$ or, equivalently, $f^{-1}q \subseteq g^{-1}q$ for every $q \in \mathsf{Sc}\, D'$. We write $\bigvee^{\mathrm{p}} F$ for the pointwise

[1] The term "arithmetic" has also been used for a poset with this property [GHK+80].

lub of a family F of functions, defined, if it exists, by $(\bigvee^{\mathrm{p}} F)x = \bigvee\{fx \mid f \in F\}$.

Scott domains and FM-domains are closed under the pointwise-ordered continuous function space. All existing lubs in the pointwise-ordered continuous function space are taken pointwise. Function application is continuous, and the category of Scott domains and continuous functions is cartesian closed, with a full sub-ccc of FM-domains and continuous functions.

For $x \in D_{\mathrm{fin}}$ and $y \in D'_{\mathrm{fin}}$, define the *step function* $[x \Rightarrow y] : D \to D'$ by setting $[x \Rightarrow y]\, x' = y$ if $x' \in \mathsf{up}\, x$, and $[x \Rightarrow y]\, x' = \bot$ otherwise. The notation $[x \Rightarrow y]$ will imply that x and y are isolated. The isolated elements of the pointwise-ordered continuous function space are those functions which are the pointwise lubs of finitely many step functions.

3 Sequentiality

3.1 Sequential Functions on Indexed Domains

In order to be able to model sequential computation, we equip domains with a parameterized notion of *indices*, intended to formalize incremental steps of a computation. Let an *index function* for a domain D be a function $I : D \to \mathcal{P}(\mathsf{St}\, D)$ such that the following properties hold, for every $x \in D$:

- True increment: For every $r \in Ix$, $x \notin r$.
- Separation: If $x < y$ then there exists $r \in Ix$ such that $y \in r$.
- Upwards motion: If $x < y$, $r \in Ix$ and $y \notin r$ then $r \in Iy$.
- Finite origin: If $r \in I(\bigvee X)$ and X is a directed set then there exists $x_0 \in X$ such that $r \in Ix_0$. Equivalently, in an algebraic poset, if $r \in Ix$ then there exists some isolated $x_0 \leq x$ such that $r \in Ix_0$.
- Definiteness: For every $r \in Ix$, $r = \mathsf{up}(\mathsf{min}\, r)$ for the set $\mathsf{min}\, r$ of minimal elements of r.

  (This is always the case for every stable open of a dI-domain.)

An *indexed domain* E is a pair $E = (D, I)$ of a domain D and an index function I for D. When convenient we blur the distinction between an indexed domain and its underlying domain.

For $x \in E$ and $s \subseteq \mathsf{up}\, x$, we call $r \in Ix$ an index of s at x iff $s \subseteq r$. We write $I(x, s)$ for the set of indices of s at x, $I(x, s) = \{r \in Ix \mid s \subseteq r\}$.

Operationally, if the current approximation of a value v being computed is x then an index $r \in Ix$ is intended to represent a possible next step in the computation, resulting in an improved approximation by selecting among the alternatives that the index offers. A sequential computation over a domain may then be seen as a sequence of choices among alternatives posed by indices at an increasing sequence of approximations. The index function determines which sequences of approximations may be computable. It may help to think of v as an input value to a program, with the program improving its approximation x to v by a process of incremental approximation, until it has sufficient information to determine its output on input v. One may also think of a program computing a sequence of ascending approximations to some target output value.

A stable open r represents a choice between its lobes. Since an index $r \in Ix$ is definite, *i.e.*, $r = \mathsf{up}(\mathsf{min}\, r)$, the choice is represented even more concretely as a choice between the elements of $\mathsf{min}\, r$, which may be seen as competing alternative approximations to the target value v. The increment in information will be to $x \vee y$, where y is the element of $\mathsf{min}\, r$ approximating v. Since r is stable open, its minimal elements are isolated and pairwise inconsistent, so that y will be unique, if

it exists. The true increment property guarantees that $x \notin r$, and thus $x < x \vee y$. If $v \notin r$, then the computation step represented by r may be said to diverge; a program that attempts to take step r at x is undefined for input v, in the input scenario, or may not output v, in the output scenario. An index $r \in I(x,s)$ of s at x may be seen as an incremental step from current approximation x towards a choice represented by s, in that it guarantees non-divergence for $v \in \mathsf{up}\, s$.

A subset $p \subseteq E$ is *sequential open* iff it is Scott open and, for every $x \in E$, either $x \in p$ or every finite $s \subseteq p \cap \mathsf{up}\, x$ has some index at x, *i.e.*, $I(x,s) \neq \emptyset$. Write $\mathsf{Sq}\, E$ for the collection of sequential opens of E, ordered by set inclusion. A function $f : E \to E'$ is sequential iff $f^{-1} q \in \mathsf{Sq}\, E$ for every $q \in \mathsf{Sq}\, E'$. Let $E \to^{\mathrm{sq}} E'$ be the sequential function space between E and E', ordered pointwise. Thanks to the generalized topological definition, it is trivial to check that the identity functions are sequential and that composition preserves sequentiality, so that indexed domains and sequential functions form a category (for any underlying class of domains).

3.2 Sequentiality in Terms of Critical Sets

By the separation property of indices, $I(x,s)$ is non-empty whenever $x < \wedge s$, so it is only interesting to ask if $I(x,s)$ is empty in the case where $x = \wedge s$. This gives rise to a definition of critical sets, which provide convenient alternative characterizations of sequentiality.

A *critical* set of an indexed domain $E = (D, I)$ is a non-empty finite subset $s \subseteq E$ that has no index at its meet, *i.e.*, such that $I(\wedge s, s) = \emptyset$. We summarize some of the development related to critical sets:

Proposition 3.1

(1) Every finite set s with a least element, and, in particular, every singleton, is critical.

(2) A set p is sequential open iff it is Scott open and closed under critical meets. For every $x \in E_{\mathrm{fin}}$, $\mathsf{up}\, x$ is sequential open. Sequential opens have the T_0 separation property.

(3) A finite set s is critical iff every sequential open that contains it also contains its meet $\wedge s$.

(4) A function $f : E \to E'$ is sequential iff it is continuous and it preserves criticality and meets of critical sets, i.e., for every critical set s of E, $fs = \{fx \mid x \in s\}$ is critical, and $f(\wedge s) = \wedge(fs)$.

(5) Every finite bounded set is critical. Every sequential open is stable open. Every sequential function is stable.

3.3 Product of indexed domains

It seems reasonable to assume that an incremental step of a sequential computation in a product domain $D_1 \times D_2$ corresponds to an increment in one of the components, but not both. This leads us to define the indexed domain product of $E_1 = (D_1, I_1)$ and $E_2 = (D_2, I_2)$ to be the indexed domain $E_1 \times E_2 = (D_1 \times D_2, I_\times)$, where $D_1 \times D_2$ is the usual domain product, ordered componentwise, and $I_\times$ is defined by

$$\begin{aligned} I_\times(x,y) \;=\; & \{r \times \mathsf{up}\, z \mid r \in I_1 x \;\&\; z \text{ is an isolated approximation of } y\} \cup \\ & \{\mathsf{up}\, z \times r \mid r \in I_2 y \;\&\; z \text{ is an isolated approximation of } x\}\,. \end{aligned}$$

It is easy to check that $I_\times$ is an index function.

Proposition 3.2 *A finite set $s \subseteq E_1 \times E_2$ is critical iff both $\pi_1\, s$ and $\pi_2\, s$ are critical.*

An indexed domain product is, in fact, a categorical product in the category of indexed domains and sequential functions (for any underlying class of domains closed under product).

Proposition 3.3 *Indexed domain product is a categorical product.*

Proof: It is sufficient and easy to show that the projections $\pi_i : E_1 \times E_2 \to E_i$ are sequential, for $i = 1, 2$, and that for sequential functions $f_i : E \to E_i$, $i = 1, 2$, the mediating morphism $\lambda x \in E\,.\,(f_1 x, f_2 x)$ is a sequential function from E to $E_1 \times E_2$. ∎

A sequential function on an indexed domain product remains sequential when one of its arguments is fixed.

Proposition 3.4 *For every sequential function $f : E_1 \times E_2 \to E'$ and every $x \in E_1$, the function* curry $f x = \lambda y \in E_2\,.\, f(x, y)$ *is a sequential function from E_2 to E'.*

3.4 Ordering the sequential function space

An adaptation of the development in [BG92] shows that FM-domains are closed under the pointwise-ordered sequential function space. In other words, if E and E' are indexed FM-domains then the sequential function space $E \to^{\mathrm{sq}} E'$, equipped with the pointwise order, is an FM-domain. Property FM is essential for this. An even simpler development shows that dI-domains are closed in the same sense under the stably-ordered function space — this is an easy corollary of the downwards closure of sequential functions in the stably-ordered stable function space.

Proposition 3.5 *FM-domains are closed under the pointwise-ordered sequential function space, regardless of the index structures used. Directed lubs and finite meets are taken pointwise, and the isolated elements are the sequential functions that are the pointwise lubs of finitely many step functions.*

3.5 Application is not sequential

Function application $\mathtt{app} : (E \to^{\mathrm{sq}} E', I) \times E \to E'$ is not sequential, no matter which index function I is used, and whether we employ the pointwise or stable orders. This also establishes that uncurrying does not preserve sequentiality, since function application is the uncurrying of an identity function. It is perhaps not surprising that uncurrying does not preserve sequentiality: the uncurried form of a function has a more complicated domain of definition, where more subtle interactions are possible that would prevent the uncurried form from being sequential.

The counter-example relies on the product structure and on the index domain axioms, and in particular on the criticality of a set with a least element, a corollary of the true increment property. Let $\mathtt{Bool}$ be the domain of booleans, with elements $\bot < \mathtt{T}, \mathtt{F}$. Consider the application function $\mathtt{app} : (\mathtt{Bool}^3 \to^{\mathrm{sq}} \mathtt{Bool}) \times \mathtt{Bool}^3 \to \mathtt{Bool}$, and the sets

$$\begin{aligned} s &= \{([(\mathtt{T},\mathtt{F},\bot) \Rightarrow \mathtt{T}] \vee [x \Rightarrow \mathtt{T}], x) \mid x \in t\} \\ t &= \{(\mathtt{T},\mathtt{F},\bot), (\bot,\mathtt{T},\mathtt{F}), (\mathtt{F},\bot\,\mathtt{T})\}\,. \end{aligned}$$

$\pi_1\, s$ has least element $[(\mathtt{T},\mathtt{F},\bot) \Rightarrow \mathtt{T}]$ in both the pointwise and stable orderings on the function space, and $\pi_2\, s = t$ is critical, since its projection on any of its three components has a least element

$\bot$. Thus s is critical. But $\mathsf{app}(\wedge s) = [(\mathtt{T},\mathtt{F},\bot)\Rightarrow\mathtt{T}](\bot,\bot,\bot) = \bot \neq \mathtt{T} = \wedge\{\mathtt{T}\} = \wedge(\mathsf{app}\, s)$, so that app fails to preserve a critical meet, and is therefore not sequential.

This negative result implies that we cannot use the framework presented here — the category **IFM** of indexed FM-domains and sequential functions — to give a sequential model for all of PCF, since application is definable in PCF (up to currying). Nevertheless, we will be able to give a sequential model for an appropriately restricted subset of PCF.

4 Interpreting types as indexed domains

We look now at ways of instantiating the index structure to obtain type interpretations in the category **IFM**. We consider the simple type system generated by the grammar $\sigma ::= \rho \mid \sigma \to \sigma'$, where ρ ranges over a set of ground types.

We assume given a flat domain $\mathcal{A}[\![\rho]\!]$ for each ground type ρ. We need to choose an index function for each such $\mathcal{A}[\![\rho]\!]$ in order to interpret ρ as an indexed domain. We then intend to interpret an arrow type $\sigma \to \sigma'$ as the sequential function space between the indexed domains representing σ and σ', ordered pointwise, with a suitably chosen index structure on the function space. This raises the question of what kind of index function is appropriate for a sequential function space.

4.1 Data sequentiality

At first-order types like $\sigma \to \sigma'$, where σ and σ' are ground, the notion of sequential function defined in [BG92] is adequate. This notion of sequentiality, which we will call *data sequentiality*, coincides with the Kahn-Plotkin sequential functions when σ and σ' are restricted to concrete domains [KP78, Cur86], and coincides with the Milner and Vuillemin notions of sequentiality when the types are restricted to products of flat domains [Mil77, Vui73].

Data sequentiality is characterized in terms of index functions as follows. Although we only need to use the definitions here when D is a flat domain (since ground types are flat), it is easy to give a more general definition for dI-domains.

For a dI-domain D we define the data-index function I^{d}_D at $x \in D$ by setting

$$I^{\mathrm{d}}_D x = \{r \in \mathsf{St}\, D \mid x \notin r \;\&\; \forall y \in \mathsf{min}\, r\,.\, (x \Uparrow y \Rightarrow x \prec x \vee y)\}\,.$$

The data-index function I^{d}_D is easily seen to be an index function for a dI-domain D.

This definition of data sequentiality requires *atomicity* of the increment represented by an index, so that successive approximations to an input will form a covering chain. If atomicity is not imposed, say, if we used $I_D x = \{r \in \mathsf{St}\, D \mid x \notin r\}$ then one could, for instance, check in a single step whether an input in $\mathtt{Bool}^3$ is in $\mathsf{up}\,\{(\mathtt{T},\mathtt{F},\bot),(\bot,\mathtt{T},\mathtt{F}),(\mathtt{F},\bot,\mathtt{T})\}$. This would clearly not be appropriate for computation in a sequential language.

Data sequentiality interacts nicely with indexed domain product. By atomicity, progress cannot be made simultaneously in different components of a product, since $(x,y) \prec (x',y')$ iff either $x \prec x'$ and $y = y'$, or $x = x'$ and $y \prec y'$; this corresponds exactly to the reasoning behind the definition of product. Therefore, the index function for the product of data-indexed domains coincides with the data-index function for the product, *i.e.*,

$$(D_1 \times D_2, I^{\mathrm{d}}_{D_1 \times D_2}) = (D_1, I^{\mathrm{d}}_{D_1}) \times (D_2, I^{\mathrm{d}}_{D_2}).$$

In [BG92] we attempted to use data sequentiality uniformly for all domains, *i.e.*, to construct a sequential model in which each type is interpreted as a data-indexed domain. This corresponds to

an operational assumption that incremental computation over a function space proceeds in the same way as incremental computation over data. This assumption is reasonable in some frameworks, such as concrete domains and sequential algorithms, and the language CDS0 [BC85, Cur86]. However, this operational assumption is not appropriate for PCF, where information about a functional argument is essentially incremented by *applying* it. We thus perceive the need to employ a different, higher-order, notion of sequentiality over the functional domains, that would correspond better to PCF's operational assumptions. (See [BG92] for further discussion.)

4.2 Constant-applicative sequentiality

In order to arrive at a higher-order notion of sequentiality more closely matching PCF's operational character, we analyze the way in which information about functional inputs is obtained in PCF. This is ultimately done by applying such an argument, as a PCF variable, say $\mathtt{f}$, to an argument of appropriate type, say, a term M, with the result of the application $\mathtt{f}M$ conveying information about the input represented by $\mathtt{f}$. Call M the *prompter* of f.

For example, consider the following PCF term M_0:

$$\begin{array}{rcl} M_0 & = & \lambda \mathtt{f} : \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool} \,. \\ & & \quad \mathtt{if}(\mathtt{f}\,\mathtt{T}\,\Omega)\&(\mathtt{f}\,\mathtt{F}\,\mathtt{T})\&\neg(\mathtt{f}\,\mathtt{F}\,\mathtt{F})\,\mathtt{then}\,\mathtt{T}\,\mathtt{else}\,\Omega, \end{array}$$

where Ω is assumed to be a divergent constant of type $\mathtt{Bool}$, and $\&$ is the left-strict-and function $\lambda \mathtt{x} \,.\, \lambda \mathtt{y} \,.\, \mathtt{if}\,\mathtt{x}\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{F}$, written in infix notation. When M_0 is applied to a term M of type $\mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool}$, M_0 may be seen as successively increasing its information about its input. The result of the application is $\mathtt{T}$ precisely when the sequence of approximations is

$$\bot,\ [(\mathtt{T},\bot)\Rightarrow \mathtt{T}],\ [(\mathtt{T},\bot)\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{T})\Rightarrow \mathtt{T}],\ [(\mathtt{T},\bot)\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{T})\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{F})\Rightarrow \mathtt{F}]$$

(up to currying). Each step of the computation corresponds to an application of $\mathtt{f}$ to some prompter. Divergence of any step would imply divergence of the entire computation. The term M_0 uses only *closed* prompters: each application of $\mathtt{f}$ is to "constant" arguments.

Consider next the prompters in the following PCF terms,

$$\begin{array}{rcl} M_1 & = & \lambda \mathtt{f} : \sigma \to \sigma' \,.\, \lambda \mathtt{x} : \sigma \,.\, \mathtt{f}\mathtt{x}, \\ M_2 & = & \lambda \mathtt{f} : \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool} \,. \\ & & \quad \mathtt{if}(\mathtt{f}(\mathtt{f}\,\mathtt{T}\,\Omega)(\mathtt{f}\Omega\,\mathtt{T}))\&(\mathtt{f}\,\mathtt{T}\,\mathtt{F})\&(\mathtt{f}\,\mathtt{F}\,\mathtt{T})\&\neg(\mathtt{f}\,\mathtt{F}\,\mathtt{F})\,\mathtt{then}\,\mathtt{T}\,\mathtt{else}\,\Omega. \end{array}$$

In the first case, M_1 denotes the identity function on the type $\sigma \to \sigma'$, or, up to currying, the corresponding application function. It is not strict in the input $\mathtt{x}$, but it is strict in the input $\mathtt{f}$. The prompter $\mathtt{x}$ of $\mathtt{f}$ may be said to be *input-dependent*, in that it involves an input other than $\mathtt{f}$. In the second case, M_2 denotes the least functional that maps the left-strict-or function $\mathsf{lor} = [(\mathtt{T},\bot)\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{T})\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{F})\Rightarrow \mathtt{F}]$ and the right-strict-or function $\mathsf{ror} = [(\bot,\mathtt{T})\Rightarrow \mathtt{T}] \vee [(\mathtt{T},\mathtt{F})\Rightarrow \mathtt{T}] \vee [(\mathtt{F},\mathtt{F})\Rightarrow \mathtt{F}]$ to $\mathtt{T}$. It is defined using *imbrication* [BCL85, p. 129]; the prompter $\mathtt{f}\,\mathtt{T}\,\Omega$ may be said to be *self-dependent*, since it uses the input $\mathtt{f}$ about which information is being sought.

We are not yet able to give a satisfactory treatment of dependent prompters. Instead, in this paper we make the simplifying assumption that prompters must be constant, *i.e.*, independent of the input. On the syntactic side, we will impose a restriction on the use of application so that we need only consider PCF terms using closed prompters. The terms M_1 and M_2 are thus excluded from consideration. On the semantic side, we assume that a computation of a value f over a

function space $E \to^{\mathrm{sq}} E'$ proceeds at each step by determining result of applying f to a constant element in E. This gives rise to the notion of *constant-applicative sequentiality*.

Corresponding to a value $x \in E_{\mathrm{fin}}$ and a "residual" index r' in E', we define a ca-index $[x \Rightarrow r']$ in the pointwise-ordered sequential function space $E \to^{\mathrm{sq}} E'$ between E and E' to be the stable open

$$[x \Rightarrow r'] = \mathsf{up}\,\{[x \Rightarrow y] \mid y \in r' \cap E'_{\mathrm{fin}}\} = \mathsf{up}\,\{[x \Rightarrow y] \mid y \in \mathsf{min}\, r'\}$$

of $E \to^{\mathrm{sq}} E'$; and we define the ca-index function $I^{\mathrm{ca}}_{E,E'}$ on $E \to^{\mathrm{sq}} E'$ by:

$$I^{\mathrm{ca}}_{E,E'} f \quad = \quad \{[x \Rightarrow r'] \mid x \in E_{\mathrm{fin}} \;\&\; r' \in I'(fx)\}\,.$$

It is easy to check that $I^{\mathrm{ca}}_{E,E'}$ is an index function for $E \to^{\mathrm{sq}} E'$. From this point on, we will assume that the sequential function space is equipped with the ca-index function; the following results depend on this choice.

Proposition 4.1 *A finite set $s \subseteq E \to^{\mathrm{sq}} E'$ is critical iff for all $x \in E_{\mathrm{fin}}$, $sx = \{fx \mid f \in s\}$ is critical.*

Proposition 4.2 *Currying preserves sequentiality, i.e., if $f : E_1 \times E_2 \to E'$ is a sequential function then $\mathsf{curry}\, f : E_1 \to (E_1 \to^{\mathrm{sq}} E')$ is a sequential function.*

Application of a fixed sequential function $f \in E \to^{\mathrm{sq}} E'$, *i.e.*, the function $\lambda z \in E\,.\, fz$, coincides with f and is therefore sequential. More importantly, application to a fixed argument is sequential.

Proposition 4.3 *For every $z \in E$, the function $\lambda f \in E \to^{\mathrm{sq}} E'\,.\, fz$ is sequential.*

Proof: If s is a critical set of $E \to^{\mathrm{sq}} E'$ then sz is critical, and $\wedge(sz) = (\wedge s)z$. ∎

4.3 Maximal uncurrying

As we have indicated, the meaning $[\![\sigma \to \sigma']\!]$ of an arrow type in the model will essentially be taken to be the sequential function space between $[\![\sigma]\!]$ and $[\![\sigma']\!]$. A further refinement is still needed. Type interpretations are usually defined in ccc's, where there is an isomorphism

$$[\![\sigma_1]\!] \to ([\![\sigma_2]\!] \to [\![\sigma']\!]) \cong ([\![\sigma_1]\!] \times [\![\sigma_2]\!]) \to [\![\sigma']\!]$$

via currying and uncurrying. In that case, it doesn't really matter which of the two is taken to define $[\![\sigma_1 \to (\sigma_2 \to \sigma')]\!]$. This is not so in our case, since uncurrying does not preserve sequentiality.

The question now arises whether a function that is sequential in its curried form, but not in its uncurried form, should be included in the sequential model. For example consider the parallel-or function, $\mathsf{por} = [(\mathtt{T}, \bot) \Rightarrow \mathtt{T}] \vee [(\bot, \mathtt{T}) \Rightarrow \mathtt{T}] \vee [(\mathtt{F}, \mathtt{F}) \Rightarrow \mathtt{F}]$. Its curried form, $\mathsf{curry\,por} : \mathtt{Bool} \to (\mathtt{Bool} \to \mathtt{Bool})$, is sequential because of the trivial index structure of $\mathtt{Bool}$. However, por itself, of type $\mathtt{Bool} \times \mathtt{Bool} \to \mathtt{Bool}$, is not sequential: $\mathsf{por}^{-1}\,\{\mathtt{T}\} = \mathsf{up}\,\{(\mathtt{T}, \bot), (\bot, \mathtt{T})\}$ is not sequential open. Moreover, parallel-or is not definable in PCF, and it is therefore desirable to exclude it from any sequential model. Thus, we will regard as "truly" sequential only those functions whose maximally uncurried form is sequential. To build a model including only such functions we will interpret arrow types in their maximally uncurried form.

4.4 The sequential type interpretation

We now define the sequential type interpretation $\mathcal{C}[\![-]\!]$, mapping each type σ to an indexed domain $\mathcal{C}[\![\sigma]\!]$:

- For a ground type ρ, $\mathcal{C}[\![\rho]\!]$ is the flat domain $\mathcal{A}[\![\rho]\!]$, equipped with the standard data-index structure.
- Each arrow type σ can be written uniquely in the form $\sigma_1 \to \cdots \to \sigma_n \to \rho$, where $n \geq 1$ and ρ is ground. We define $\mathcal{C}[\![\sigma]\!]$ to be the sequential function space $\mathcal{C}[\![\sigma_1]\!] \times \cdots \times \mathcal{C}[\![\sigma_n]\!] \to^{\mathrm{sq}} \mathcal{C}[\![\rho]\!]$, ordered pointwise, with the standard constant-applicative index structure.

We assume that we have at least ground types `Bool` and `Nat`, corresponding to the the usual flat domains of truth values and natural numbers respectively.

5 A sub-language of PCF

5.1 The ca-PCF typing system and semantics

Raw (untyped) terms are built from a given set of constants, identifiers, application and abstraction in the usual way, as in PCF. We define axioms and inference rules for judgements of the form $\Gamma \vdash M : \sigma$, to be read as: the term M has type σ in type context Γ. A type context is a finite ordered list of identifier-type pairs, and we write $\Gamma, v : \sigma$ for the type context obtained by extending Γ with the binding $v : \sigma$. Identifiers may occur more than once in a type environment, and the rightmost occurrence always takes precedence. The essential restriction imposed by our typing system is that a variable of functional type may only be applied to closed terms. This captures the simplifying assumption that prompters cannot depend on the input. For convenience we also require that a variable of functional type be applied successively to as many arguments as needed to obtain a result of ground type; this restriction is not important.

The terms of ca-PCF are those terms M for which a judgement $\Gamma \vdash M : \sigma$ is derivable. We use L to range over terms, and K to range over closed terms. A term K is closed iff it has no free identifiers; equivalently, if $\vdash K : \sigma$ is derivable for some σ.

We define a semantic function $\mathcal{C}[\![-]\!]$ for judgements $\Gamma \vdash M : \sigma$ by induction on the proof of the judgement. Throughout we assume that Γ has form $v_1 : \gamma_1, \ldots, v_m : \gamma_m$ and that σ is written in the form $\sigma_1 \to \cdots \to \sigma_n \to \rho$, where ρ is ground. The meaning of $\Gamma \vdash M : \sigma$ will be

$$\begin{array}{rcl} \mathcal{C}[\![\Gamma \vdash M : \sigma]\!] & \in & \mathcal{C}[\![\gamma_1 \to \cdots \to \gamma_m \to \sigma]\!] \\ & = & \mathcal{C}[\![\gamma_1]\!] \times \cdots \times \mathcal{C}[\![\gamma_m]\!] \times \mathcal{C}[\![\sigma_1]\!] \times \cdots \times \mathcal{C}[\![\sigma_n]\!] \to^{\mathrm{sq}} \mathcal{C}[\![\rho]\!]. \end{array}$$

Note that the environment is "blended into" the semantic domains; this is necessary, since all functions in the model, including the meanings of terms, are to be fully uncurried.

We assume a semantic function $\mathcal{A}[\![-]\!]$ for constants such that $\mathcal{A}[\![c]\!] \in \mathcal{A}[\![\sigma]\!]$ for each constant c of type σ. As in PCF we assume at least the following constants with their usual interpretations.

- A constant of type ρ for each element of each ground type ρ. In particular,
  - A numeral of type `Nat` for each natural number.
  - Constants `T` and `F` of type `Bool`, denoting the corresponding truth values.
  - For each ground type ρ a constant Ω^ρ of that type, denoting the least element of $\mathcal{A}[\![\rho]\!]$.

- For each ground type ρ, a constant $\mathtt{if}^\rho$ of type $\mathtt{Bool} \to \rho \to \rho \to \rho$ such that

$$\mathcal{A}[\![\mathtt{if}^\rho]\!] = \bigvee\nolimits^{\mathrm{p}} \{[(\mathtt{T}, x, \bot) \Rightarrow x], [(\mathtt{F}, \bot, x) \Rightarrow x] \mid x \in \mathcal{A}[\![\rho]\!]\} .$$

- Arithmetic constants: $(+1)$ and (-1) of type $\mathtt{Nat} \to \mathtt{Nat}$, $(=0)$ of type $\mathtt{Nat} \to \mathtt{Bool}$, denoting the successor and predecessor functions and the equal-to-zero predicate, respectively. As in PCF, $\mathcal{A}[\![(-1)]\!]0 = \bot$.

- Basic operations on ground types other than $\mathtt{Bool}$ and $\mathtt{Nat}$ are left unspecified. We will later assume existence of constants necessary for definability.

The following are the axioms and inference rules for the typing system, together with the definition of the semantic function $\mathcal{C}[\![-]\!]$.

- Constants:

$$\frac{}{\Gamma \vdash c : \sigma}\ \mathsf{const}$$

$$\begin{aligned}\mathcal{C}[\![\Gamma \vdash c : \sigma]\!] \quad = \quad & \lambda(x_1 \in \mathcal{C}[\![\gamma_1]\!], \ldots x_m \in \mathcal{C}[\![\gamma_m]\!], y_1 \in \mathcal{C}[\![\sigma_1]\!], \ldots, y_n \in \mathcal{C}[\![\sigma_n]\!]) . \\ & \mathcal{A}[\![c]\!](y_1, \ldots, y_n)\end{aligned}$$

  for every constant c of type σ.

- Variables:

$$\frac{\vdash K_1 : \sigma_1 \quad \ldots \quad \vdash K_n : \sigma_n}{\Gamma \vdash v K_1 \ldots K_n : \rho}\ \mathsf{ca\text{-}var}$$

$$\begin{aligned}\mathcal{C}[\![\Gamma \vdash v K_1 \ldots K_n : \rho]\!] \quad = \quad & \lambda(x_1 \in \mathcal{C}[\![\gamma_1]\!], \ldots, x_m \in \mathcal{C}[\![\gamma_m]\!]) . \\ & x_i(\mathcal{C}[\![\vdash K_1 : \sigma_1]\!], \ldots, \mathcal{C}[\![\vdash K_n : \sigma_n]\!])\end{aligned}$$

  provided i is the rightmost position in Γ of an occurrence of v, and $\sigma = \gamma_i$.

  For a variable of ground type, $n = 0$, this specializes to the familiar variable introduction rule:

$$\frac{}{\Gamma \vdash v : \rho}\ \mathsf{var}_0$$

$$\mathcal{C}[\![\Gamma \vdash v : \rho]\!] = \lambda(x_1 \in \mathcal{C}[\![\gamma_1]\!], \ldots, x_m \in \mathcal{C}[\![\gamma_m]\!]) . x_i$$

  provided i is the rightmost position in Γ of an occurrence of v, and $\rho = \gamma_i$.

- Application:

$$\frac{\Gamma \vdash L : \sigma_0 \to \sigma \quad \Gamma \vdash L_0 : \sigma_0}{\Gamma \vdash L L_0 : \sigma}\ \mathsf{app}$$

$$\begin{aligned}\mathcal{C}[\![\Gamma \vdash L L_0 : \sigma]\!] \quad = \quad & \lambda(x_1 \in \mathcal{C}[\![\gamma_1]\!], \ldots, x_m \in \mathcal{C}[\![\gamma_m]\!], y_1 \in \mathcal{C}[\![\sigma_1]\!], \ldots, y_n \in \mathcal{C}[\![\sigma_n]\!]) . \\ & \mathcal{C}[\![\Gamma \vdash L : \sigma_0 \to \sigma]\!](x_1, \ldots, x_m, f(x_1, \ldots, x_m), y_1, \ldots, y_n)\end{aligned}$$

where $\sigma_0 = \sigma_0^1 \to \cdots \to \sigma_0^{n_0} \to \rho_0$ and

$$\begin{array}{rcl} f & = & \lambda(x_1 \in \mathcal{C}[\![\gamma_1]\!], \ldots, x_m \in \mathcal{C}[\![\gamma_m]\!]) \,. \\ & & \quad \lambda(z_1 \in \mathcal{C}[\![\sigma_0^1]\!], \ldots, z_{n_0} \in \mathcal{C}[\![\sigma_0^{n_0}]\!]) \,. \, \mathcal{C}[\![\Gamma \vdash L_0 : \sigma_0]\!](x_1, \ldots, x_m, z_1, \ldots, z_{n_0}). \end{array}$$

- Abstraction:

$$\frac{\Gamma, v : \sigma \vdash L : \sigma'}{\Gamma \vdash (\lambda v : \sigma \,.\, L) : \sigma \to \sigma'}\,\mathsf{abs}$$

$$\mathcal{C}[\![\Gamma \vdash (\lambda v : \sigma \,.\, L) : \sigma \to \sigma']\!] = \mathcal{C}[\![\Gamma, v : \sigma \vdash L : \sigma']\!]$$

Proposition 5.1 *Every term has a unique type: if* $\Gamma \vdash L : \sigma$ *and* $\Gamma \vdash L : \sigma'$ *are both derivable then* $\sigma = \sigma'$.

The semantic function $\mathcal{C}[\![-]\!]$ *is well-defined, and for every derivable judgement* $\Gamma \vdash L : \sigma$,

$$\mathcal{C}[\![\Gamma \vdash L : \sigma]\!] \in \mathcal{C}[\![\gamma_1 \to \cdots \to \gamma_m \to \sigma]\!],$$

where $\Gamma = v_1 : \gamma_1, \ldots, v_m : \gamma_m$.

5.2 Definability of isolated elements

The link between the syntactic restrictions of ca-PCF and the semantic assumptions of the sequential model is formulated as a full abstraction result.

Proposition 5.2 *For every type* σ *and each isolated* $x \in \mathcal{C}[\![\sigma]\!]$ *there exists a closed term* Def_x *such that* $\mathcal{C}[\![\vdash \mathrm{Def}_x : \sigma]\!] = x$.

Moreover, for every ground type ρ', $k > 0$ *and each finite sequence* $X = x_1, \ldots, x_k$ *of isolated elements of* $\mathcal{C}[\![\sigma]\!]$, *if* $\mathtt{up}\, X$ *is an index at* x *then there is a closed term* Sel_X *such that*

$$\begin{array}{l} \mathcal{C}[\![\vdash \mathrm{Sel}_X : \sigma \to (\rho')^k \to \rho']\!] = \\ \quad \lambda(z \in \mathcal{C}[\![\sigma]\!], y_1 \in \mathcal{C}[\![\rho']\!], \ldots, y_k \in \mathcal{C}[\![\rho']\!]) \,. \, (\vee^{\mathrm{P}} \{[x_i \Rightarrow y_i] \mid i \leq k\}) z. \end{array}$$

We call such a term a selector for X.

Proof: By type induction on σ.

If σ is a ground type we have already assumed the existence of the relevant defining constants.

We can choose for $\mathrm{Sel}_{\mathtt{T},\mathtt{F}} : \mathtt{Bool} \to \rho' \to \rho' \to \rho'$ the constant $\mathtt{if}^{\rho'}$. For other selectors over $\mathtt{Bool}$ use the obvious variations.

For $\mathrm{Sel}_{x_1,\ldots,x_k} : \mathtt{Nat} \to (\rho')^k \to \rho'$ take

$$\begin{array}{rcl} \mathrm{Sel}_{x_1,\ldots,x_k} & = & \lambda \mathtt{z} : \mathtt{Nat} \,.\, \lambda \mathtt{y}_1 : \rho' \,. \ldots . \lambda \mathtt{y}_k : \rho' \,. \\ & & \quad \mathtt{if}\, \mathtt{z} = 0\, \mathtt{then}\, M_0\, \mathtt{else} \\ & & \qquad \mathtt{if}\, \mathtt{z} = 1\, \mathtt{then}\, M_1\, \mathtt{else} \\ & & \qquad\quad \ldots \\ & & \qquad\quad \mathtt{if}\, \mathtt{z} = k'\, \mathtt{then}\, M_{k'}\, \mathtt{else} \\ & & \qquad\qquad \Omega \end{array}$$

where $k' = \mathsf{max}\{x_1, \ldots, x_k\}$, and for $0 \leq j \leq k'$, $\mathtt{z} = j$ is short for $(=0)((-1)^j\,\mathtt{z})$, and $M_j = \mathtt{y}_i$ if there exists i such that $j = x_i$, and $M_j = \Omega$ otherwise. It is important that the testing of $\mathtt{z}$ against the values $0, \ldots, k'$ be carried out in increasing order.

For other ground types we need to assume the existence of appropriately interpreted constants to allow a similar definition of selector terms.

If σ is not ground, assume that $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \rho$, and let f be an isolated sequential function in $\mathcal{C}[\![\sigma]\!]$. Since f is isolated, it is the lub of a finite set of step functions. Choose a minimal set F of step functions such that $f = \bigvee F$, say $F = \{[\bar{x}_i \Rightarrow y_i] \mid i \leq l\}$, where each $\bar{x}_i = (x_i^1, \ldots, x_i^n)$. By minimality of l, each of the y_i's is a proper value. Continue now by induction on l.

If $l = 0$ then $f = \bot$, so we can let $\mathrm{Def}_f = \Omega^\sigma = \lambda \mathtt{v}_1 : \sigma_1 \ldots . \lambda \mathtt{v}_n : \sigma_n \,.\, \Omega^\rho$.

If $l = 1$ then $f = [(x_1^1, \ldots, x_1^n) \Rightarrow y_1]$. By the induction hypothesis there are closed terms Def_{y_1} and selectors $\mathrm{Sel}_{x_1^j}$ for each j. We can take

$$\mathrm{Def}_f = \lambda \mathtt{v}_1 : \sigma_1 \ldots . \lambda \mathtt{v}_n : \sigma_n \,.\, \mathrm{Sel}_{x_1^1}\, \mathtt{v}_1 (\ldots (\mathrm{Sel}_{x_1^n}\, \mathtt{v}_n (\mathrm{Def}_{y_1})) \ldots).$$

If $l > 1$, let $s = \{\bar{x}_i \mid i \leq l\}$. Clearly, $\mathsf{up}\, s = f^{-1}(\mathcal{C}[\![\rho]\!] \setminus \{\bot\})$. By minimality of F, s has no least element, or else f would be a single step function, given that $\mathcal{C}[\![\rho]\!]$ is a flat domain. Therefore $\wedge s \notin \mathsf{up}\, s$. By sequentiality of f, $\mathsf{up}\, s$ is sequential open, so that s is not critical. It has an index at $\wedge s$ in the product $\mathcal{C}[\![\sigma_1]\!] \times \cdots \times \mathcal{C}[\![\sigma_n]\!]$, which is derived from an index in one of the components; assume without loss of generality that it is derived from an index in the m'th component, so that there is an $r \in I(\wedge(\pi_m\, s), \pi_m\, s)$. If we take a minimal r (with respect to number of lobes) it will have at most l lobes, by minimality, but at least 2 lobes, since $\wedge s \notin r = \mathsf{up}(\mathsf{min}\, r)$, using definiteness. Let l' be the number of lobes of r, so that $r = \mathsf{up}\,\{z_j \mid j \leq l'\}$. This now lets us split F into corresponding collections of step functions, each with less than l elements, that may be distinguished on the basis of r. More formally, for $j \leq l'$, let $f_j = \bigvee F_j$, where $F_j = \{[\bar{x}_i \Rightarrow y_i] \mid i \leq l \;\&\; z_j \leq x_i^m\}$. Since each f_j is the lub of less than l step functions, it is definable, by induction hypothesis. We are now able to define f:

$$\mathrm{Def}_f = \lambda \mathtt{v}_1 : \sigma_1 \ldots . \lambda \mathtt{v}_n : \sigma_n \,.\, \mathrm{Sel}_{z_1, \ldots z_{l'}}\, \mathtt{v}_m (\mathrm{Def}_{f_1}\, \mathtt{v}_1 \ldots \mathtt{v}_n) \ldots (\mathrm{Def}_{f_{l'}}\, \mathtt{v}_1 \ldots \mathtt{v}_n).$$

We now show definability of $\mathrm{Sel}_{f_1, \ldots, f_k}$ in the functional case. If $\mathsf{up}\,\{f_i \mid i \leq k\}$ is an index at f in $\mathcal{C}[\![\sigma]\!]$ then there must exist $\bar{x}_0 = (x_0^1, \ldots, x_0^n)$ such that $f_i = [\bar{x}_0 \Rightarrow y_i]$, and $\{y_i \mid i \leq k\}$ is an index at $f\bar{x}_0$ in $\mathcal{C}[\![\rho]\!]$. We are therefore able to transform the selection problem in the function space into a selection problem in the ground case, which has already been solved. We thus obtain:

$$\begin{aligned} \mathrm{Sel}_{f_1, \ldots, f_k} \quad = \quad & \lambda \mathtt{f} : \sigma \,.\, \lambda \mathtt{v}_1 : \rho' \ldots . \lambda \mathtt{v}_k : \rho' \,. \\ & \mathrm{Sel}_{y_1, \ldots, y_k} (\mathtt{f}\, \mathrm{Def}_{x_0^1}, \ldots, \mathrm{Def}_{x_0^n}) \mathtt{v}_1 \ldots \mathtt{v}_k . \end{aligned}$$

Note that $\mathtt{f}$ is applied to closed arguments, so this is a valid term. ∎

The essential difference between this definability proof and Plotkin's proof for the parallel extension of PCF [Plo77, lemma 4.5] is in the synthesis of the defining term for arrow type with $l > 1$, in the above terminology. Plotkin's proof uses the parallel conditional facility to combine a

defining term for the lub of l step functions with an additional step function to obtain a defining term for the lub of $l+1$ step functions; we rely instead on the existence of an index that partitions the set of step functions into smaller sets.

Full abstraction — both inequational and equational — follows by standard arguments from the definability of isolated elements.

Proposition 5.3 *The semantics $\mathcal{C}[\![-]\!]$ is inequationally fully abstract with respect to itself as a notion of program behavior. That is, for any pair of derivable judgements $\Gamma \vdash L : \sigma$ and $\Gamma \vdash L' : \sigma$,*

$$\mathcal{C}[\![\Gamma \vdash L : \sigma]\!] \leq \mathcal{C}[\![\Gamma \vdash L' : \sigma]\!]$$

iff, for every appropriate[2] program context $P[-]$ of type ρ,

$$\mathcal{C}[\![\vdash P[L] : \rho]\!] \leq \mathcal{C}[\![\vdash P[L'] : \rho]\!].$$

To link up this result with the standard notion of behavior for PCF programs, we verify that $\mathcal{C}[\![-]\!]$ agrees with the usual operational semantics for PCF, as presented in [Plo77].

Proposition 5.4 *The program behaviors induced by the semantics $\mathcal{C}[\![-]\!]$ and the operational semantics coincide. That is, for every closed term P of ground type ρ, $\mathcal{C}[\![\vdash P : \rho]\!] = x \neq \perp$ iff P evaluates to (the constant denoting) x, and $\mathcal{C}[\![\vdash P : \rho]\!] = \perp$ iff the evaluation of P diverges.*

In summary, the semantics $\mathcal{C}[\![-]\!]$ is fully abstract for ca-PCF with respect to the usual notion of program behavior.

5.3 Recursive definitions

Since the fixpoint operator is continuous but not sequential in our framework, we cannot simply add the usual fixpoint constants $\mathtt{Y}$ to the language. Instead, we can introduce μ-abstraction, so that $\mu\mathtt{f} : \sigma \,.\, M\mathtt{f}$ is equivalent to $\mathtt{Y}M$. However, we then need to relax the term-forming syntactic constraints to allow μ-bound variables to be applied to input arguments inside the body M, so as to permit non-trivial uses of recursion, e.g. the term

$$(\times 2) = \mu\mathtt{f} : \mathtt{Nat} \to \mathtt{Nat} \,.\, \lambda\mathtt{x} : \mathtt{Nat} \,.\, \mathtt{if}\ \mathtt{x} = 0\ \mathtt{then}\ 0\ \mathtt{else}(\mathtt{f}(\mathtt{x} - 1) + 2),$$

where the recursively defined variable $\mathtt{f}$ has an input-dependent prompter. The meaning of every term is in the right semantic domain when supplied with appropriate values for its free μ-bound variables.

6 Conclusion

We have introduced a notion of indexed domain and shown that it permits a general definition of sequential function enjoying certain domain-theoretic properties. In particular, we obtain a class of indexed domains containing the flat domains, closed under product, and closed under the pointwise-ordered sequential function space. We have shown that a particular kind of index structure on function spaces gives rise to a fully abstract semantics for a non-trivial sub-language of PCF. Nevertheless, unrestricted application is not a sequential function in our model, and it

[2] Since we do not associate fixed types with variables we must assign to holes in program contexts a type context Γ which they provide, as well as the type of the term that they expect in the hole.

remains to be seen if we can find a yet more sophisticated notion of index structure that would cope satisfactorily with full PCF. This would have to deal with the complications caused by imbrication and what we have called input- or self-dependent prompters. The generalized indices should, like the indices presented here, have a firm operational grounding, and they should carry information that can be used for showing definability of the sequential functions in the generalized framework.

There are interesting connections and significant differences with the work of Bucciarelli and Ehrhard [BE91]. The critical sets of an indexed domain always form a coherence structure in the sense of Bucciarelli and Ehrhard (and the sequential functions in our model correspond to their strongly stable functions). The converse is not true, because our requirements on index structures are stronger, so as to build in the ability to model incremental computation. Bucciarelli and Ehrhard also use data sequentiality at ground types, and essentially the same product. They obtained a cartesian closed category of strongly stable functions between qualitative domains equipped with coherence structure, using the stable ordering on function spaces; in particular, in their model application is sequential with respect to the *stable* ordering. However, the coherence structures that they use on function types do not correspond to index structures, and apparently do not convey enough operational information to model incremental sequential computation. Moreover, the *pointwise* ordering is of primary relevance for the PCF full abstraction problem, since it corresponds to the operational pre-order on terms of function type, and therefore we are more concerned to find a notion of sequential function space using the pointwise order.

References

[BC82] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[BC85] G. Berry and P.-L. Curien. Theory and practice of sequential algorithms: the kernel of the applicative language CDS0. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 2, pages 35–87. Cambridge University Press, 1985.

[BCL85] G. Berry, P.-L. Curien, and J.-J. Lévy. Full abstraction for sequential languages: the state of the art. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 3, pages 89–132. Cambridge University Press, 1985.

[BE91] A. Bucciarelli and T. Ehrhard. Sequentiality and strong stability. In *Proc. Sixth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, July 1991.

[Ber78] G. Berry. Stable models of typed λ-calculi. In *Proc. 5^{th} Coll. on Automata, Languages and Programming*, number 62 in Lecture Notes in Computer Science, pages 72–89. Springer-Verlag, July 1978.

[BG92] S. Brookes and S. Geva. Stable and sequential functions on Scott domains. Technical Report CMU-CS-92-121, School of Computer Science, Carnegie Mellon University, June 1992.

[CF92] R. Cartwright and M. Felleisen. Observable sequentiality and full abstraction. In *Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pages 328–342. ACM Press, January 1992.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Research Notes in Theoretical Computer Science. Pitman, 1986. Second edition, expanded and updated, published by Birkhäuser, Boston, 1993.

[Cur92] P.-L. Curien. Observable algorithms on concrete data structures. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 432–443. IEEE Computer Society Press, June 1992.

[GHK+80] G. Gierz, K.H. Hoffman, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *A Compendium of Continuous Lattices*. Springer Verlag, 1980.

[KP78] G. Kahn and G. D. Plotkin. Domaines concrets. Rapport 336, IRIA-LABORIA, 1978. English translation (with historical introduction by S. Brookes) to appear in *Theoretical Computer Science*, 1993.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[Mul87] K. Mulmuley. *Full Abstraction and Semantic Equivalence*. MIT Press, 1987.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[Vui73] J. Vuillemin. *Proof techniques for recursive programs*. PhD thesis, Stanford University, 1973.

[Zha91] G. Q. Zhang. *Logic of Domains*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.